Markt&Technik

Wilfried Häring

AMIGA DOS 1.3

Anwender- und Programmiererbuch

Bedienung der Shell (CLI) ★ DOS-Referenzteil ★ Batchprogrammierung ★ FastFileSystem ★ Linker ★ DOS-Editoren ★ MEMACS ★ Programmierschnittstellen zu Amiga-DOS ★ Terminal-Ein- und Ausgabe ★ Binärdateien ★ Dateiformate

AMIGA DOS 1.3

Wilfried Häring

AMIGA DOS 1.3

Anwender- und Programmiererbuch

Bedienung der Shell (CLI) • DOS-Referenzteil • Batchprogrammierung • FastFileSystem • Linker • DOS-Editoren • MEMACS • Programmierschnittstellen zu Amiga-DOS • Terminal-Ein- und Ausgabe • Binärdateien • Dateiformate

Markt&TechnikVerlag AG

CIP-Titelaufnahme der Deutschen Bibliothek

Häring, Wilfried:
AMIGA-DOS 1.3 : Anwender- und Programmiererbuch /
Wilfried Häring. –
Haar bei München : Markt-u.-Technik-Verl., 1989
ISBN 3-89090-802-0

Die Informationen in diesem Produkt werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht.
Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.
Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen.
Trotzdem können Fehler nicht vollständig ausgeschlossen werden.
Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.
Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien.
Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Commodore-Amiga ist eine Produktbezeichnung der Commodore-Amiga Inc., USA.
Amiga-BASIC ist eine Produktbezeichnung der Microsoft Inc., USA.
AmigaDOS ist eine Produktbezeichnung der Commodore-Amiga Inc., USA.

Der Abdruck der korrekten Tabellen zum Aufzeichnungsformat des
alten Filing-System geschah mit freundlicher Genehmigung von
Ralph Babel aus seiner Publikation »Discovery – Der Diskeditor«, 1988.

15 14 13 12 11 10 9 8 7 6 5 4 3
92 91 90

ISBN 3-89090-802-0

© 1989 by Markt&Technik Verlag Aktiengesellschaft,
Hans-Pinsel-Straße 2, D-8013 Haar bei München/West-Germany
Alle Rechte vorbehalten
Einbandgestaltung: Grafikdesign Heinz Rauner
Dieses Produkt wurde mit Desktop-Publishing-Programmen erstellt
und auf der Linotronic 300 belichtet
Druck: Wiener Verlag, Himberg
Printed in Austria

Inhaltsverzeichnis

I. Anwender-Buch

Vorwort 15

1 Einleitung 17

Ein erster Kontakt mit AmigaDOS

1.1 Einleitung 17
1.1.1 Die Aktivierung der SHELL 17
1.1.2 Der Gebrauch der SHELL 17
1.1.3 Startdateien 17
1.1.4 Workbench und SHELL – Verwandtschaft und Unterschiede 18
1.2 Unterschiede zwischen Workbench 1.2 und 1.3 19

2 Amiga-DOS stellt sich vor 21

Die ersten Schritte und Erfahrungen mit Amiga-DOS

2.1 Überblick 21
2.2 Editiermöglichkeiten in einem SHELL-Fenster 22
2.2.1 Schreibung der DOS-Befehle 23
2.3 Alles über Dateien 23
2.3.1 Die Dateinamen 24
2.3.2 Verzeichnisse anlegen 25
2.3.3 Setzen des aktuellen Verzeichnisses 26
2.3.4 Benennung des aktuellen Gerätes 28
2.3.5 Vereinbarungen logischer Geräte 29
2.3.6 Die RAM-Disks 31
2.4 Theorie und Praxis der Amiga-DOS-Befehle 31
2.4.1 Befehle im Hintergrund ausführen lassen 32
2.4.2 Batchprogrammierung mit Script-Dateien 32
2.4.3 Ein- und Ausgabeumleitung von Befehlen 33
2.4.4 Unterbrechung eines Programms 33
2.4.5 Das Befehlsformat 33
2.5 Der Aktualisierungsprozeß 35
2.6 Die gebräuchlichsten Befehle: Eine kleine Übersicht 36

2.6.1 Einführung in einige Amiga-DOS-Anweisungen 36
2.6.2 Zu Beginn 36
2.6.3 Eine Diskette kopieren 37
2.6.4 Eine Diskette formatieren 37
2.6.5 Eine Diskette bootfähig machen 38
2.6.6 Umbenennen einer Diskette 38
2.6.7 Das Inhaltsverzeichnis einer Diskette lesen 39
2.6.8 Benutzung des LIST-Befehls 40
2.6.9 Schützen einer Datei 40
2.6.10 Informationen über eine Diskette lesen 41
2.6.11 Wechsel des aktuellen Verzeichnisses 41
2.6.12 Datum und Uhrzeit neu stellen 42
2.6.13 Umleiten der Ausgabe in eine Datei 42
2.6.14 Eine Textdatei auf dem Bildschirm ausgeben 42
2.6.15 Umbenennen einer Datei 43
2.6.16 Eine Datei löschen 43
2.6.17 Kopieren von Dateien mit zwei Laufwerken 44
2.6.18 Dateien mit einem Laufwerk kopieren 44
2.6.19 Ein neues Verzeichnis einrichten 45
2.6.20 Anweisungen beim Systemstart automatisch durchführen lassen 45
2.6.21 Logische Geräte erzeugen, ändern und löschen 45
2.6.22 Ein neues SHELL-Fenster öffnen 47
2.6.23 Ein SHELL-Fenster schließen 47
2.6.24 Zum Schluß dieses Abschnittes 48

3 Die Amiga-DOS-Befehle 49

Vollständige Dokumentation der DOS-Befehle, Stand Workbench 1.3 Revision 34.20.

3.0 Verwendete Fachausdrücke 49
3.0.1 Nach folgendem Schema werden die Befehle dokumentiert 50
3.1 Die SHELL 51
3.1.1 ESCAPE-Sequenzen und CTRL-Codes 53
3.1.2 Editieren von Befehlszeilen 54
3.1.3 Befehls-Makros 54
3.2 Die DOS-Befehle 55

4 Batchprogrammierung 113

Erläuterung des Erstellens von Script-Dateien an Beispielen und Analyse der Startup-Sequence

4.0 Einführung in Script-(Batch)-Dateien 113
4.1 Starten und Parameterübergabe 114
4.2 DOS-Befehle zur Steuerung von Script-Dateien 122
4.3 Ein Spezialfall: Die Startup-Sequence 129

4.3.1 Eine eigene Startup-Sequence 132
4.3.2 Booten von Fast-Filing-System 134
4.3.3 Einrichten einer Bootpartition für A2090A 135
4.3.4 Booten einer nicht autobootfähigen Festplatte von Diskette 136
4.3.5 Booten von der RAD: 137
4.3.6 Anlegen einer nicht bootfähigen RAD: 139
4.4 Patternmatching mit SPAT und DPAT 140
4.5 Push und Pop für das aktuelle Verzeichnis 142

5 Die Mountlist 144

Beschreibung der einzelnen Felder eines Mountlisteintrags und Analyse der Beispieleinträge der Workbench 1.3

6 Der Makro-Assembler 153

Beschreibung des Standard-Makroassembler für den Amiga

6.1 Einführung zum Mikroprozessor MC 68000 153
6.2 Der Aufruf des Assemblers 155
6.3 Aufbau des Quellcodes 156
6.3.1 Kommentare 156
6.3.2 Ausführbare Instruktionen 156
6.3.2.1 Das Label-Feld 156
6.3.2.2 Lokale Labels 157
6.3.2.3 Das Opcode-Feld 157
6.3.2.4 Das Operanden-Feld 158
6.3.2.5 Das Kommentar-Feld 158
6.4 Ausdrücke 158
6.4.1 Operatoren 158
6.4.2 Operand/Operator-Kombinationen 158
6.4.3 Symbole 159
6.4.4 Zahlen 160
6.5 Adressierungs-Arten 160
6.6 Verschiedene Befehlstypen 161
6.7 Direktiven 162
6.7.1 Direktiven zur Assembler-Kontrolle 164
6.7.2 Direktiven zur Symbol-Definition 165
6.7.3 Direktiven zur Daten-Definition 166
6.7.4 Listing-Kontrolle 167
6.7.5 Bedingte Assemblierung 169
6.7.6 Direktiven zu Makros 170
6.7.7 Externe Symbole 172
6.7.8 Allgemeine Direktiven 172
6.7.9 Die Objekt-File-Struktur 173
6.7.10 hunk_unit (999/3E7) 174

6.7.11 hunk_name (1000/3E8) 175
6.7.12 hunk_code (1001/3E9) 175
6.7.13 hunk_data (1002/3EA) 175
6.7.14 hunk_bss (1003/3EB) 176
6.7.15 hunk_reloc32 (1004/3EC) 176
6.7.16 hunk_reloc16 (1005/3ED) 177
6.7.17 hunk_reloc8 (1006/3EE) 177
6.7.18 hunk_ext (1007/3EF) 177
6.7.19 hunk_symbol (1008/3E0) 179
6.7.20 hunk_debug (1009/3F1) 180
6.7.21 hunk_end (1010/3F2) 180

7 Der Linker ALINK 181

Beschreibung des Standard-Systemlinkers des Amiga

7.0 Terminologie 181
7.1 Einführung 183
7.2 Zur Anwendung des Linkers 185
7.2.1 Syntax der Befehlszeile 185
7.2.2 WITH-Files 186
7.2.3 Fehler und andere Ausnahmebehandlungen 187
7.2.4 MAP und XREF 188
7.3 Overlays 189
7.3.1 Die OVERLAY-Direktive 189
7.3.2 Symbol-Referenzen 191
7.3.3 Warnungen und Hinweise 192
7.4 Weiterführende Hinweise 193
7.4.1 Überblick über die Hunk-Overlay-Tabelle 193
7.4.1.1 Einen Overlay-Baum planen 194
7.4.1.2 Beschreibung des Baums 195
7.5 Ladbare Dateien (Load-Files) 196
7.5.1 hunk_header (1011/3F3) 197
7.5.2 hunk_overlay (1013/3F5) 198
7.5.3 hunk_break (1014/3F6) 198
7.6 Beispiele 198
7.7 Fehlercodes und -meldungen 202
7.8 ATOM (Alink Temporary Object Modifier) 203
7.8.1 Das »Problem« 203
7.8.2 Die bisherige Lösung 203
7.8.3 Die Lösung durch ATOM 203
7.8.4 Zukünftige Lösungen 204
7.8.5 Das bewirken die neuen Bits 204
7.8.6 Wirkung der Neuerung 204
7.8.9 Erforderliche Software 205

7.8.10 Syntax der ATOM-Anweisung 205
7.8.11 Beispiele für ATOM-Aufrufe 206
7.8.12 Fehlermeldungen 207

8 ED – Der Bildschirm-Editor 209

Dieses Kapitel beschreibt den Bildschirm-Editor ED

8.1 Einführung in den Bildschirm-Editor 209
8.2 Unmittelbare Befehle 210
8.2.1 Die Benutzung des Cursors 210
8.2.2 Text einfügen 211
8.2.3 Löschen von Text 212
8.2.4 Scrollen des Textes 213
8.2.5 Wiederholung von Anweisungen 213
8.3 Erweiterte Anweisungen 213
8.3.1 Programmsteuerung 214
8.3.2 Blocksteuerung 215
8.3.3 Verändern der Cursorposition 216
8.3.4 Suchen und Ersetzen von Text 217
8.3.5 Text ändern 218
8.3.6 Wiederholte Befehle 218
8.3.7 Kurzübersicht der ED-Befehle 219

9 EDIT – Der Zeileneditor 222

Zeileneditor zum programmgesteuerten Bearbeiten von Textdateien

9.1 EDIT – Ein zeilenorientierter Texteditor 222
9.1.1 Aufrufen von EDIT 222
9.1.2 EDIT-Kommandos einsetzen 224
9.1.2.1 Die aktuelle Zeile 224
9.1.2.2 Zeilennummern 224
9.1.2.3 Auswahl der aktuellen Zeile 225
9.1.2.4 Qualifikatoren 227
9.1.2.5 Ändern der aktuellen Zeile 227
9.1.2.6 Löschen ganzer Zeilen 228
9.1.2.7 Einfügen neuer Zeilen 229
9.1.2.8 Wiederholen von Befehlen 230
9.1.3 Verlassen von EDIT 230
9.1.4 Ein kombiniertes Beispiel: alle Befehle auf einmal 231
9.2 Vollständige Beschreibung des Zeileneditors 232
9.2.1 Befehlssyntax 233
9.2.1.1 Befehlsnamen 233
9.2.1.2 Argumente 233
9.2.1.3 Zeichenketten 234

9.2.1.4	Mehrfach-Zeichenketten	234
9.2.1.5	Qualifizierte Zeichenketten	234
9.2.1.6	Suchausdrücke	235
9.2.1.7	Zahlen	235
9.2.1.8	Einstellwerte	235
9.2.1.9	Befehlsgruppen	235
9.2.1.10	Wiederholung der Befehle	236
9.2.2	Programmablauf	236
9.2.2.1	Das Eingabe-Zeichen (Prompt)	236
9.2.2.2	Die aktuelle Zeile	237
9.2.2.3	Zeilennummern	237
9.2.2.4	Qualifizierte Zeichenketten	237
9.2.2.5	Abspeichern der editierten Zeilen	238
9.2.2.6	Handhabung des Dateiendes	238
9.2.3	Funktionelle Einteilung der EDIT-Befehle	238
9.2.3.1	Auswahl einer aktuellen Zeile	239
9.2.3.2	Einfügen und Löschen einer Zeile	240
9.2.4	Zeilenfenster	241
9.2.4.1	Das operationale Fenster	241
9.2.4.2	Verändern einzelner Zeichen in der aktuellen Zeile	242
9.2.5	Verändern von Zeichenketten in der aktuellen Zeile	243
9.2.5.1	Grundlegende Operationen mit Zeichenketten	243
9.2.5.2	Der Null-String	243
9.2.5.3	Die Zeiger-Variante	244
9.2.5.4	Löschen von Teilen der aktuellen Zeile	244
9.2.6	Weitere Befehle für die aktuelle Zeile	245
9.2.6.1	Trennen und Verbinden von Zeilen	245
9.2.7	Teile der Quelldatei lesen	246
9.2.8	Verwaltung von Befehls-, Eingabe- und Ausgabedateien	247
9.2.8.1	Befehlsdateien	248
9.2.8.2	Eingabedateien	248
9.2.8.3	Ausgabedateien	249
9.2.9	Schleifen	250
9.2.10	Umfassende Operationen	250
9.2.10.1	Festlegen von umfassenden Änderungen	251
9.2.10.2	Aufheben von umfassenden Änderungen	251
9.2.10.3	Aufschub umfassender Änderungen	251
9.2.11	Anzeigen des Programmstatus	252
9.2.12	Beenden eines EDIT-Ablaufs	252
9.2.13	Verifizieren der aktuellen Zeile	252
9.2.14	Verschiedenartige Befehle	253
9.2.15	Abbrechen der interaktiven Editierung	254
9.3	Kurzübersicht	254

10 Memacs 258

Der Standardeditor MicroEmacs auf dem Amiga

10.1 Einführung in das Editieren mit MEMACS 259
10.2 Das Project-Menü 260
10.3 Das Edit-Menü 261
10.4 Das Window-Menü 264
10.5 Das Move-Menü 265
10.6 Das Line-Menü 266
10.7 Das Word-Menü 267
10.8 Das Search-Menü 268
10.9 Das Extras-Menü 268
10.10 MEMACS-Befehle, die nur über Tastatureingeben aktiviert werden können. 270
10.11 Anlegen einer Konfigurationsdatei von MEMACS 271

Anhang Druckertreiber 1.3 272

II. Programmierer-Buch

1 Einleitung 288

Erster Kontakt mit Amiga-DOS für Programmierer

1.1 Eine Übersicht über die Systemsoftware 289
1.2 Eine Übersicht über die Hardware des Amiga 291
1.2.1 Der MC68k 291
1.2.2 Die Customchips 292
1.2.3 Die Trennung zwischen CHIP- und FAST-RAM 293
1.2.4 Der Kickstart 293
1.2.5 Die Timerbausteine 294

2 Programmierrichtlinien für das Programmieren in C 295

Ein Vorschlag für das Layout von C-Quelltexten

2.1 Formatierung von C-Programmen 295
2.2 Einpassen in das Betriebssystem 299
2.3 Parameterübergabe beim Start eines Programms 301

3 Die Amiga-DOS-Datenstrukturen 310

Beschreibung der Datenstrukturen, die Amiga-DOS verwendet

3.0 Einführung 310
3.1 Datenstrukturen für die Prozeßverwaltung 311

3.2 Globale Datenstrukturen 315
3.2.1 Die Info-Substruktur 316
3.3 Speicherverwaltung 319
3.4 Segment-Listen 319
3.5 Datei-Handles 320
3.6 Locks 321

4 DOS-Bibliotheksfunktionen 325

Die Funktionsbibliothek für den Programmierer alphabetisch sortiert

5 Packets 344

Direkte Steuerung von Handlerprozessen

5.0 Beschreibung der Struktur DosPacket 344
5.1 Packet-Typen 347

6. Das Amiga-DOS-File-System 354

Das Aufzeichnungsformat des alten Filing-System

6.1 Die Amiga-DOS-File-Struktur 354
6.1.1 Das Wurzelverzeichnis einer Diskette (Root-Block) 354
6.1.2 Benutzerverzeichnisblock 355
6.1.3 Der Dateikopf-Block 356
6.1.4 Der Datei-List-Block 357
6.1.5 Der Datenblock 357
6.1.6 Die Bitmap 358
6.1.7 Der Bootblock (Diskettentyp und Bootvorgang) 358

7 Terminal-Ein- und -Ausgabe auf dem Amiga 359

7.0 Einführung 359
7.0.1 Hilfreiche Amiga-DOS-Befehle 360
7.1 CON:-Tastatur-Eingabe 360
7.2 CON:-Bildschirm-Ausgabe 362
7.3 RAW:-Bildschirm-Ausgabe 362
7.4 RAW:-Tastatur-Eingabe 366
7.5 Auswahl von RAW:-Ereignissen 367

Anhang Listings 374

Stichwortverzeichnis 383

Anwender-Buch

Vorwort

Dieses Buch richtet sich an zwei Gruppen von Lesern, die nicht unbedingt leicht unter einen Hut zu bringen sind.

Der Computeranwender findet hier einen systematischen Einstieg in die Bedienung und Nutzung des Betriebssystems, um sich seine Arbeitsumgebung auf dem Amiga maßzuschneidern. Er lernt die Bedienung der SHELL und alle DOS-Befehle kennen und versteht, wie Erweiterungen (zum Beispiel Festplatten) in das System eingebunden werden. Außerdem werden die drei Standardtexteditoren für den Amiga beschrieben, ED, EDIT und MEMACS, sowie der Systemlinker und -Assembler. Das Buch beschäftigt sich nicht mit der Hardware des Rechners.

Der Programmierer findet eine vollständige Dokumentation der Programmierschnittstelle zum Amiga-DOS. Die für die Sprache C kommentierten Datenstrukturen, die vom DOS genutzt werden, und eine alphabetisch geordnete Erklärung aller dem Programmierer zur Verfügung stehenden DOS-Funktionsaufrufe. Dieses Buch beschäftigt sich nicht mit dem Multitasking-Kern EXEC des Betriebssystems. Für Programmbeispiele wurde die Version 5.02 des Lattice C-Compiler verwendet, der den C ANSI-Standard weitgehend unterstützt, und dem Programmierer eine mächtige Programmierumgebung an die Hand gibt.

Als Vorlage wurde die amerikanische Originaldokumentation verwendet (Amiga-DOS-Anwender-Handbuch, Amiga-DOS-Programmierer-Handbuch und das Technische Amiga-DOS-Handbuch). Durch Einschluß aller Neuerungen zur Betriebssystemversion 1.3 und eine völlige Neustrukturierung, Erweiterung und Überarbeitung des alten Amiga-DOS-Handbuch 1.2, entstand das jetzt vorliegende Handbuch zur Version 1.3 des Betriebssystems.

Da das Entstehen eines Buches nicht nur vom Autor allein getragen wird, möchte ich mich bei allen bedanken, die direkt oder indirekt zu diesem Buch beigetragen haben, und besonders bei folgenden Personen:

Herrn Rudolph Kühnert, der mich erst in die Lage brachte, dieses Buch zu schreiben.

Ralph Babel, dem nimmer müde werdenden Quell von Denkanstößen und Ideen für die Tabellen zur Dateistruktur des alten Filing-Systems.

Meinem Freund Mark Dörr für das Lesen des Manuskripts und seine wertvolle Hilfe.

Wilfried Häring

1 Einleitung

1.1 Einleitung

Dieser erste Teil des Handbuchs beschreibt das Amiga-DOS und dessen Befehlsschatz aus der Sicht eines Anwenders, der die SHELL oder das CLI, die alte Version der SHELL, benutzt, um Dateien und Programme zu verwalten.

Amiga-DOS ist das auf Disketten oder eine Festplatte gestützte Betriebssystem Ihres Amiga. Die SHELL-(dtsch. Muschelschale)Benutzer-Schnittstelle liest Ihre Tastatureingaben (Amiga-DOS-Anweisungen) ein, stellt sie im SHELL-Fenster dar und übergibt sie dem Amiga-DOS, um sie vom Rechner ausführen zu lassen.

In dieser Hinsicht ist die SHELL eine herkömmliche Computer-Schnittstelle: Es werden Anweisungen über die Tastatur eingegeben, der Kommandozeilen-Interpreter zeigt sie auf dem Bildschirm an und macht sie für den Computer verständlich, dann werden sie ausgeführt.

Die Workbench dagegen ist für den normalen Anwender aufgrund der Eingabe von Anweisungen über die Maus und grafischen Symbolen (Icons) und Menüleisten angenehm leicht zu bedienen, in den Möglichkeiten schränkt sie den Anwender jedoch leider etwas ein.

1.1.1 Die Aktivierung der SHELL

Starten Sie Ihren Amiga mit der Workbench-Diskette (falls Sie einen Amiga 1000 besitzen, müssen Sie diesen zuvor natürlich auch die Kickstart-Diskette einlesen lassen). Als erstes öffnen Sie dann das Disketten-Icon der Workbench-Diskette durch zweimaliges Anklikken. Jetzt öffnet sich ein Fenster, das den Inhalt dieser Diskette anzeigt. Das SHELL-Icon wird daraufhin sichtbar und kann durch einen Doppelklick mit der Maus geöffnet werden.

1.1.2 Der Gebrauch der SHELL

Wenn die SHELL nach anklicken mit der Maus das aktive Fenster ist, können Sie nun die gewünschten Anweisungen über die Tastatur eingeben. SHELL-Fenster können wie andere Fenster geformt und bewegt werden. Verwenden Sie zum Schließen des SHELL-Fensters einfach den Befehl ENDCLI.

1.1.3 Startdateien

Wird die SHELL oder das CLI durch Doppelklicken auf das entsprechende Icon (Piktogramm) geöffnet, führt das Hilfsprogramm die Befehle der jeweiligen Startdatei aus. Für die SHELL ist dies die Datei »S:SHELL-Startup«, für den CLI die Datei »S:CLI-Startup«. Beide Startdateien befinden sich im Verzeichnis *S:* der Workbench-Diskette und sollten sich auf Ihrer Boot-Diskette oder Festplatte ebenfalls dort befinden.

1.1.4 Workbench und SHELL – Verwandtschaft und Unterschiede

Unter dem Begriff Verzeichnis (Directory) verstehen wir das Inhaltsverzeichnis aller Dateien auf einer Diskette. Ein Verzeichnis kann ein oder mehrere Unterverzeichnisse beinhalten, die ein oder mehrere Dateien unter einem Namen zusammenfassen. Mit der Anweisung DIR erscheint diese Liste der Dateien und Unterverzeichnisse der augenblicklich benutzten Diskette auf dem Bildschirm; wenn Sie noch die Workbench eingelegt haben, eine Liste der Dateien, die auf der Workbench-Diskette vorhanden sind.

Die Dateien und Directories entsprechen weitgehend den auf der Workbench sichtbaren Icons. Geben Sie also direkt in das soeben geöffnete SHELL-Fenster den Befehl DIR ein. Wenn Sie das mit DIR erschienene Verzeichnis genauer anschauen, werden Sie feststellen, daß sich in diesem Verzeichnis mehr Dateien befinden, als Icons im Workbench-Fenster sichtbar sind. Die Workbench zeigt, im Unterschied zur SHELL nur dann eine Datei »Harribald« an, wenn eine mit ihr verknüpfte Datei »Harribald.info« existiert. Die Workbench verwendet die .info-Datei, um das Icon auf dem Bildschirm darzustellen und mit einer entsprechenden Datei zu verknüpfen.

Auf der Workbench-Diskette in der Schublade (grafische Repräsentation für ein Unterverzeichnis) »System« befindet sich zum Beispiel das Programm DiskCopy. Es besteht aus zwei Dateien. Die Datei »DiskCopy« enthält das Kopierprogramm und »DiskCopy.info« die Information darüber, wie das Icon aussieht.

Für den Amiga existiert zum Beispiel das Malprogramm Deluxe Paint II. Nehmen wir an, mit diesem Programm wurde eine Bildschirmgrafik erstellt und unter dem Namen »Mein_Erstes.pic« abgespeichert. Für die Datei »Mein_Erstes.pic« erzeugt das Programm nun ein Projekt-Icon, das bedeutet, daß die Datei Daten enthält, die mit einem anderen Programm erzeugt wurden.

Das Aussehen des Icons und die Information, daß »Mein_Erstes.pic« mit Deluxe Paint II erstellt wurde, werden in der Datei »Mein_Erstes.pic.info« abgelegt. Wenn Sie die Bilddatei »Mein_Erstes.pic« angeklickt und geöffnet haben, lädt die Workbench zuerst das Malprogramm Deluxe Paint II und teilt Deluxe Paint II mit, daß das Bild »Mein_Erstes.pic« angezeigt werden soll.

Ähnliches passiert auch mit Basic-Programmen, die Sie mit Amiga-Basic erstellt und abgespeichert haben.

Es besteht ein Zusammenhang zwischen den in der Workbench verwendeten Schubladen-Icons und den Verzeichnissen und Unterverzeichnissen unter SHELL. Die Schubladen-Icons repräsentieren ein Unterverzeichnis. Das Aufrufen eines Verzeichnisses entspricht dem Anklicken eines Schubladen-Icons. Geräte, die direkt auf gespeicherte Dateien auf einem Speichermedium zugreifen können, wie die Diskettenlaufwerke (DF0:) oder eine Festplatte (DH0:), werden ebenfalls über ein Icon, das Disketten-Icon, angesprochen. Obwohl beide Schnittstellen, Workbench und SHELL, mit Amiga-DOS zusammenarbeiten, unterscheiden sie sich so weit voneinander, daß nicht alle Programme oder Anweisungen der SHELL auch auf der Workbench verwendet werden können. Die in Kapitel 2 dieses Handbuches beschriebenen Anweisungen können nur unter der SHELL benutzt werden.

1.2 Unterschiede zwischen Workbench 1.2 und 1.3

Einer der deutlichsten Unterschiede zwischen den Workbench-Versionen 1.2 und 1.3 ist die Erweiterung der Workbench um das Programm SHELL. Der Hauptunterschied zwischen CLI und SHELL ist, daß die SHELL eine erweiterte CLI-Umgebung bietet. Das bedeutet, daß die SHELL die volle Leistung des CLI und deutliche Erweiterungen besitzt. Da SHELL und CLI viele Gemeinsamkeiten aufweisen, wird der Begriff SHELL synonym für CLI und SHELL benutzt.

2 Amiga-DOS stellt sich vor

Dieses Kapitel gibt einen Überblick über das Amiga-DOS-Betriebssystem, beschreibt die Handhabung der SHELL-Fenster und die Hierarchie der Verzeichnisse (Directories) und zeigt den Gebrauch der DOS-Anweisungen. Am Ende des Kapitels finden Sie einfache Übungsbeispiele zum Einsatz der Amiga-DOS-Anweisungen.

2.1 Überblick

Amiga-DOS ist ein Multitasking-Betriebssystem, das auf der Grundlage des Betriebssystems TRIPOS entwickelt wurde. Multitasking bedeutet, daß gleichzeitig mehrere Programme ablaufen können. Obwohl TRIPOS als Mehrplatz-Betriebssystem entwickelt worden ist, wird Amiga-DOS in der Regel für den Einzelplatz-Einsatz gebraucht, wobei es dann mit seiner Multitasking-Fähigkeit Vordergrund- und Hintergrundprogramme quasi simultan erledigt.

Jeder einzelne Arbeitsablauf des Betriebssystems, zum Beispiel die Dateiverwaltung, aber auch jedes laufende Programm wird Prozeß genannt. Es wird jeweils nur ein Prozeß zur selben Zeit ausgeführt. Während dieser Zeit warten andere Programme (das sind natürlich ebenfalls Prozesse) auf ihren Aufruf oder, nach einer Unterbrechung, auf ihre Fortsetzung. Jeder Prozeß erhält eine Priorität. Je nach Priorität verteilt sich dann die Rechenzeit. Je größer die Priorität ist, desto mehr Rechenzeit bekommt der betroffene Prozeß zugeteilt.

Das DOS des Amiga benutzt eine Reihe von Prozessen, die den normalen Anwender unterstützen, indem sie im Hintergrund Verwaltungsaufgaben erfüllen, zum Beispiel die Steuerung der Diskettenlaufwerke, der Festplatte und der Druckerschnittstelle und die Verwaltung der seriellen Datenübertragung. Diese Prozesse werden auch private Prozesse des Betriebssystems genannt. Sie sind eigentlich lauter kleine eigenständige Programme.

Einer der Prozesse des Amiga-DOS ist die Benutzerschnittstelle SHELL. Sie können sowohl mehrere SHELL-Fenster gleichzeitig öffnen und ihnen unterschiedliche Prioritäten zuweisen, als auch innerhalb eines SHELL-Fensters mehrere Hintergrundprozesse starten, die von 1 bis n durchnumeriert werden. Dafür können die Befehle NEWCLI, NEWSHELL oder RUN verwendet werden. Ein SHELL-Fenster schließen Sie mit der Anweisung ENDCLI. Die genannten Anweisungen werden in Kapitel 3 dieses Handbuches ausführlich behandelt.

2.2 Editiermöglichkeiten in einem SHELL-Fenster

Über die Tastatur eingegebene Anweisungen werden von der SHELL an den Kommandozeilen-Interpreter gerichtet, der dann das Amiga-DOS veranlaßt, das entsprechende Werkzeugprogramm zu starten oder mit einer Fehlermeldung zurückzukehren.

Anweisungen können aber auch an ein laufendes Programm gerichtet werden, das das SHELL-Fenster für seine Ein- und Ausgaben benutzt. In diesem Fall führt der Console-Handler »NEWCON:«, der alte »CON:« oder vielleicht der aus der Public Domain stammende »ConMan« die Ein- und Ausgabe durch, je nachdem, welcher dieser Console-Handler installiert wurde. Die Original-Workbench 1.3 installiert automatisch den »NEWCON:«-handler, die Workbench 1.2 nutzt den »CON:« handler. Diese SHELL-Fenster-Kontrolle und Steuerung hilft dem Benutzer bei der Zeileneditierung und führt verschiedene andere Funktionen aus.

Sie können bis zu 255 Zeichen pro Zeile eingeben. Mit der Taste [BACKSPACE] können Sie eventuelle Tippfehler verbessern. Ein Druck auf diese Taste löscht das zuletzt eingegebene Zeichen. Geben Sie bei gedrückter Control-Taste ([Ctrl]) den Buchstaben »x« ein, so wird die Zeile, in der sich der Cursor befindet, gelöscht. Solche Control-Kombinationen werden im weiteren mit [Ctrl]+[X] bezeichnet.

Sobald irgend etwas eingegeben wird, stoppt Amiga-DOS jede Ausgabe auf dem Bildschirm, bis die Eingabe (durch Betätigung der Taste [RETURN]) abgeschlossen wurde, um eine verwirrende Mischung von Ein- und Ausgabetexten zu verhindern. Amiga-DOS erkennt das Ende einer Anweisungszeile am Drücken der Return-Taste oder am Löschen einer Zeile mit [Ctrl]+[X] beziehungsweise [BACKSPACE], das so lange gedrückt bleiben muß, bis alle Zeichen der Zeile gelöscht sind. Sobald Amiga-DOS erkannt hat, daß die Eingabe beendet wurde, wird die zurückgehaltene Ausgabe fortgesetzt. Die Ausgabe können Sie durch Drücken irgendeiner Taste (die [Space]-Taste dürfte am einfachsten sein) anhalten. Um die Ausgabe wieder aufzunehmen, müssen entweder die Backspace-, die Return-Taste oder [Ctrl]+[X] betätigt werden.

Die Tastenkombination [Ctrl]+[/] interpretiert das Amiga-DOS als Dateiende-Zeichen. In verschiedenen Fällen wird durch diese Kombination eine Eingabedatei geschlossen. Diese Anweisungen werden später genauer behandelt.

Wenn fremdartige Zeichen auf dem Bildschirm erscheinen, ohne daß Sie etwas auf der Tastatur eingegeben haben, haben Sie sicher versehentlich [Ctrl]+[O] gedrückt. Das DOS des Amiga sieht diese Kombination als eine Anweisung für den Console-Handler an, den alternativen Zeichensatz zu verwenden. Dies können Sie durch [Ctrl]+[N] wieder rückgängig machen. Alle bisherigen Zeichen sollten nun wieder normal erscheinen. Es ist auch möglich, den Bildschirm mit [Esc]+[C] zu löschen und wieder normalen Text ausgeben zu lassen. Achtung: Bei der Eingabe über den Console-Handler CON: funktionieren weder die Funktions- noch die Cursortasten.

Der Handler NEWCON: ist ein leistungsfähiger Ersatz für den Handler CON: und wird durch den Namen NEWCON: angesprochen. Ein NEWCON:-Fenster bietet dem Benutzer wesentlich mehr Flexibilität und Komfort bei der Befehlseingabe. Im Gegensatz zu CON:, das zum Korrigieren von Schreibfehlern nur [BACKSPACE] zuläßt, d.h. alle Zeichen bis zum Fehler müssen gelöscht werden, um danach den Text neu einzugeben, erlaubt ein NEWCON:-Fenster das Ändern der eingegebenen Befehlszeile(n). Im Gegensatz zum CLI benutzt die SHELL ein NEWCON:-Fenster. Nachstehend eine Liste der Fähigkeiten von NEWCON:-Fenstern:

Die Pfeiltasten [Cursor nach links] und [Cursor nach rechts] können verwendet werden, um den Cursor in der Befehlszeile zu bewegen. Die Taste [BACKSPACE] löscht das Zeichen links vom Cursor, die Taste [DELETE] das Zeichen unter dem Cursor. [Ctrl]+[Y] löscht Text ab der Cursorposition bis zum Zeilenende. [Ctrl]+[U] löscht Text ab der Cursorposition bis zum Zeilenanfang. [Ctrl]+[W] setzt den Cursor an die nächste Tabulatorposition. Die Tastenkombination [Shift]+[Cursor nach links] oder ([Ctrl]+[A]) bewegt den Cursor zum Anfang der Zeile. Durch [Shift]+[Cursor nach rechts] wird er zum Zeilenende bewegt.

NEWCON: verfügt über eine »commandline history«, einen zirkularen »Kommandozeilenspeicher« mit 2 Kbyte Kapazität. Im Lauf des Arbeitens eingegebene Befehle können durch Verwendung der Taste [Cursor nach oben] wieder aufgerufen werden. (Die Taste [Cursor-nach unten] bewegt den Cursor in Vorwärtsrichtung durch den Befehlspuffer. Dies ist nützlich, wenn Sie unachtsam weiter als bis zum gesuchten Befehl zurückgeblättert haben.)

Die Tastenkombination [Shift]+[Cursor nach oben] oder ([Ctrl]+[R]) sucht in Rückwärtsrichtung nach einer als Muster teilweise eingegebenen Zeichenfolge. Die Tastenkombination [Shift]+[Cursor nach unten] bewegt den Cursor zum Ende des Befehlspuffers.

Steuerzeichen werden unter NEWCON: durch inverse Zeichen (Steuerzeichencode + 64) dargestellt. Die Zeichen werden als Steuerzeichen gelesen und können wie bisher in Dateien verwendet werden.

2.2.1 Schreibung der DOS-Befehle

Amiga-DOS erkennt alle eingegebenen Anweisungen und Argumente in Groß- und/oder Kleinschrift. Das DOS des Amiga zeigt einen Dateinamen in der Form an, in der er beim Speichern der Datei gebildet wurde. Im Gegensatz dazu wird er vom DOS in jeder Form gefunden, unabhängig, welche Schreibweise Sie zum Bezeichnen des Dateinamens beim erneuten Öffnen verwenden.

2.3 Alles über Dateien

Dieser Abschnitt beschreibt die Amiga-DOS-Dateiverwaltung. Im einzelnen erklärt er, wie Dateien benannt, organisiert und aufgerufen werden. Eine Datei (File) ist die kleinste benannte Dateneinheit, die von Amiga-DOS benutzt wird. Die einfachste Identifikation einer Datei wird durch die Angabe des Dateinamens erreicht, wie in Abschnitt 2.3.1 beschrieben wird. Unter Umständen kann es nötig sein, eine Datei genauer zu benennen.

Diese Benennung enthält dann den Device- oder Diskettennamen und/oder die Verzeichnisnamen die dem Dateinamen vorangestellt sind, den »Pfad« in dem sich die Datei befindet. Auch dies wird in den nächsten Abschnitten besprochen.

2.3.1 Die Dateinamen

Das DOS des Amiga verwaltet Daten und Informationen in einer Anzahl von Dateien auf einer Diskette oder Festplatte. Sie sollten so benannt werden, daß der Benutzer sie leicht wiedererkennen und aufrufen kann. Die Dateienverwaltung erlaubt Dateinamen, die bis zu 30 Zeichen lang sein können. Alle verfügbaren Zeichen des Amiga, außer Doppelpunkt (:) und Schrägstrich (/), sind zulässig. Das heißt, in einem Dateinamen dürfen Leerzeichen, Gleichheitszeichen, das Plus (+) und Anführungsstriche (') vorkommen (auch mehrere Punkte sind erlaubt), also alle Zeichen, die von der SHELL anerkannt werden. Nicht jedoch Sonderzeichen eines nationalen Zeichensatzes wie »ä, ö, ü, ß ,...«!

Nichtsdestotrotz muß der Dateiname in Anführungszeichen angegeben werden, wenn spezielle Zeichen (Leerzeichen, Anführungsstrich, Schrägstrich) in einem Dateinamen verwendet werden. Sollen Anführungsstriche im Dateinamen verwendet werden, so ist unmittelbar davor ein Sternchen (*) zu setzen. Ähnlich muß zur Verwendung eines Sternchens zusätzlich ein zweites Sternchen eingefügt werden. Das bedeutet, daß Sie den Dateinamen

```
A*B = C"
```

wie folgt eingeben müssen:

```
"A**B = C*""
```

damit er von der SHELL korrekt angenommen wird.

Vorsicht! Der Gebrauch des Sternchens in dieser Weise widerspricht vielen anderen Betriebssystemen, wo es allgemein als Joker verwendet wird. Dieser Joker bewirkt dort, daß bei Eingabe von *Dat** alle Dateien, die mit diesen drei Zeichen beginnen, ausgewählt werden.

Wird unter Amiga-DOS ein Sternchen (*) mit Leerzeichen von weiteren Zeichen getrennt eingegeben, so repräsentiert es die Tastatur und das aktuelle Fenster als Standard-Ein- und Ausgabe!

Zum Beispiel kopiert

```
COPY Name TO *
```

die Datei »Name« auf den Bildschirm.

Leerzeichen sollten Sie am Anfang und am Ende eines Dateinamens vermeiden, da sie leicht zu Mißverständnissen führen. Als Grundregel: Benutzen sie markige, griffige und einfache Namen, denn Sie wollen sich sicher nicht jedesmal die Finger abbrechen und das Gehirn verrenken, wenn Sie ihre eigenen Dateien aufrufen wollen.

2.3.2 Verzeichnisse anlegen

In der Dateiverwaltung werden Verzeichnisse zum Zusammenfassen von Dateien in logische Einheiten gebraucht. So können Sie zum Beispiel zwei verschiedene Verzeichnisse dazu verwenden, um Programme von Programmerläuterungen zu trennen. Jede Datei auf einer Diskette muß in einem Verzeichnis stehen. Eine leere Diskette enthält ein Verzeichnis, das als Wurzelverzeichnis oder auch Hauptinhaltsverzeichnis bezeichnet wird. Wenn eine Datei auf eine leere Diskette geschrieben wird, dann steht sie in diesem Wurzelverzeichnis. Verzeichnisse können baumartig vom Wurzelverzeichnis aus wachsen. Sie können beliebig viele weitere Verzeichnisse beinhalten. In jedem Verzeichnis können deshalb Dateien, weitere Verzeichnisse oder beides stehen. Jeder Dateiname muß innerhalb des Verzeichnisses, in dem er steht, einzigartig sein, zwei Dateien gleichen Namens können in einem Verzeichnis nicht gleichzeitig existieren. Wird der gleiche Name für eine zweite Datei angegeben, wird die erste Datei gleichen Namens überschrieben, ihr Inhalt geht verloren!

Andererseits dürfen zwei Dateien in unterschiedlichen Verzeichnissen den selben Namen annehmen. Auf diese Weise unterscheidet sich die Datei »Martin« im Verzeichnis »Thomas« vollkommen von einer Datei mit dem Namen »Martin« im Verzeichnis »Carola«. Teilen sich zwei Anwender oder zwei Prozesse eine Diskette, so brauchen sie nicht zu befürchten, daß sie sich gegenseitig die Dateien überschreiben, solange sie in ihrem eigenen Verzeichnis bleiben.

Falls eine Datei gelöscht wird, wird darüber keine Meldung auf dem Bildschirm ausgegeben. Der Aufbau der Verzeichnisse kann ebenso zur Organisation von Informationen auf der Diskette verwendet werden. Zum Beispiel, um verschiedene Arten von Dateien in verschiedenen Verzeichnissen aufzubewahren.

Dazu ein Beispiel: Angenommen, eine Diskette enthält zwei Verzeichnisse mit Namen »Bernd« und »Monika«. Das Verzeichnis »Bernd« enthält zwei Dateien, die »Text« und »Brief« heißen. Das Verzeichnis »Monika« enthält eine Datei mit Namen »Daten« und zwei Verzeichnisse mit Namen »Brief« und »Rechnung«. Diese Unterverzeichnisse enthalten je eine Datei »Juni18«.

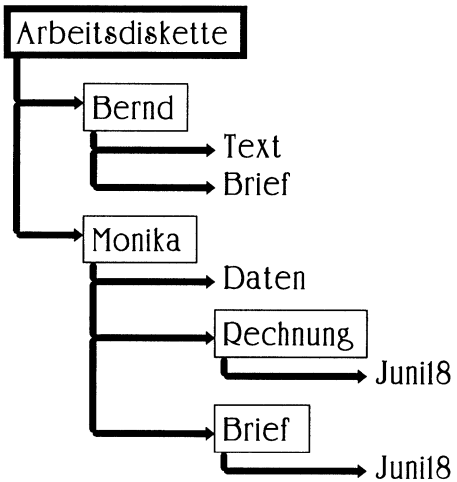


Bild 2.0: Aufbau einer Verzeichnis-Struktur

Achtung: Das Verzeichnis »Bernd« hat eine Datei mit Namen »Brief«, während das Verzeichnis »Monika« ein Verzeichnis mit Namen »Brief« besitzt. Hier kann keine Verwirrung entstehen, weil beide Strukturen sich in verschiedenen Verzeichnissen befinden. Die Verzweigung der Verzeichnisse kann nach unten beliebig fortgeführt werden. Um eine Datei vollständig zu bezeichnen, muß in den Zugriffspfad das Verzeichnis, in dem die Datei sich befindet, bis zum Wurzelverzeichnis einbezogen sein. Um eine Datei zu bezeichnen, geben Sie also die Namen aller Verzeichnisse auf dem Weg zur angestrebten Datei an. Um jeden Verzeichnisnamen vom nächsten Namen zu trennen, muß ein Schrägstrich (/) verwendet werden. Die vollständigen Bezeichnungen der Datendateien in Bild 2.0 lauten dann wie folgt:

```
Arbeitsdiskette:Bernd/Text
Arbeitsdiskette:Bernd/Brief
Arbeitsdiskette:Monika/Daten
Arbeitsdiskette:Monika/Rechnung/Juni18
Arbeitsdiskette:Monika/Brief/Juni18
```

2.3.3 Setzen des aktuellen Verzeichnisses

Es ist sehr aufwendig, eine vollständige Dateibezeichnung, wie oben beschrieben, einzugeben. Aus diesem Grunde stellt die Dateienverwaltung Ihres Amiga ein aktuelles Verzeichnis als Bezugspunkt zur Verfügung. Die Dateiverwaltung sucht in diesem aktuellen Verzeichnis nach der benötigten Datei. Um das aktuelle Verzeichnis festzulegen, verwenden Sie den Befehl CD, die Abkürzung für Change Current Directory. Wird in dem obigen Beispiel das Verzeichnis

```
Arbeitsdiskette:Monika/Rechnung
```

mit der Befehlssequenz

```
CD Arbeitsdiskette:Monika/Rechnung
```

zum aktuellen Verzeichnis gemacht, dann reicht der folgende Name zum Bezeichnen der Datei aus:

```
Juni18
```

Jedes Verzeichnis kann zum aktuellen Verzeichnis gemacht werden. Um irgendeine Datei innerhalb dieses Verzeichnisses zu bezeichnen, genügt es also, den Namen der Datei einzugeben, nachdem man vorher mit CD das Verzeichnis gewechselt hat. Um Dateien in Unterverzeichnissen des aktuellen Verzeichnisses zu bestimmen, genügt es, die Namen der Verzeichnisse vom aktuellen Verzeichnis ausgehend abwärts zu benennen. Alle anderen Dateien auf der Diskette sind noch wie üblich erreichbar, auch wenn jetzt ein aktuelles Verzeichnis gesetzt ist. Um Amiga-DOS allerdings alle Verzeichnisse vom Wurzelverzeichnis aus durchsuchen zu lassen, setzen Sie einen Doppelpunkt (:) an den Anfang des Dateinames. Wird »Monika« zum aktuellen Verzeichnis, kann die Datei »Text« im Verzeichnis »Bernd« immer noch durch Eingabe der Bezeichnung

```
:Bernd/Text
```

erreicht werden.

Das aktuelle Verzeichnis erspart auf einfache Art eine Menge Tipparbeit, weil nun statt der wiederholten Eingabe des gesamten Pfades nur der einmalige Wechsel des Verzeichnisses ausreicht.

Ein Schrägstrich (/) bezeichnet nicht das Wurzelverzeichnis wie in einigen Systemen, zum Beispiel in der Welt des Betriebssystems Unix. Er verweist vielmehr auf das Verzeichnis über dem aktuellen Verzeichnis. Amiga-DOS erlaubt die mehrfache Verwendung von Schrägstrichen. Jeder Schrägstrich verweist auf die nächst höhere Verzeichnisebene. Ist zum Beispiel »:Monika/Brief« das aktuelle Verzeichnis, können Sie die Datei »:Monika/Rechnung/Juni18« als »/Rechnung/Juni18« bezeichnen. Um die Dateien in :Bernd zu erreichen, geben Sie zuerst ein:

```
CD :Bernd
```

oder

```
CD //Bernd
```

Danach können Sie jede Datei in Bernd einfach nur mit dem Dateinamen bezeichnen. Natürlich dürfen Sie auch immer // zum direkten Zugriff auf die spezielle Datei benutzen. Zum Beispiel:

```
TYPE //Bernd/Brief
```

Dieser Ausdruck zeigt die Datei, ohne daß zuvor »Bernd« als aktuelles Verzeichnis gesetzt werden muß. Um direkt auf die Stufe des Wurzelverzeichnisses zu gelangen, müssen Sie nur einen Doppelpunkt (:) vor dem Verzeichnis-Namen eingeben.

2.3.4 Benennung des aktuellen Gerätes

Geräte (Devices) haben Namen, damit Sie unter deren Angabe auf sie verweisen können. Laufwerksbezeichnungen wie DF0: oder DF1: sind Beispiele für Device-Namen. Wie bei den Dateinamen können Device-Namen in Groß- und Kleinschrift eingegeben werden.

Ein Device ist ein physikalisches Peripheriegerät, wie Diskettenlaufwerk, Drucker, Festplatte, Tastatur oder Bildschirm, aber auch simulierte Geräte, wie RAM-Disks o.ä., die später behandelt werden, fallen darunter. Stehen Ihnen beispielsweise mehrere Diskettenlaufwerke zur Verfügung, wird jede Floppy in der Form DFn: bezeichnet. Die Variable n steht dabei für die Nummer des Laufwerks, gezählt von Null an. Zum Beispiel entspricht DF1: dem zweiten angeschlossenen Diskettenlaufwerk. Augenblicklich kann Amiga-DOS mit maximal vier Diskettenlaufwerken zusammenarbeiten. Das heißt, es sind die Device-Namen DF0: bis DF3: erlaubt.

Wie Sie wissen, wird das Wurzelverzeichnis des aktuellen Laufwerks durch einen Doppelpunkt (:) vor der Dateibeschreibung gekennzeichnet. Das Wurzelverzeichnis eines speziellen (anderen) Laufwerks oder einer Festplatte wird angegeben, indem der Doppelpunkt nach der Laufwerksbezeichnung eingetippt wird.

Um auf eine Datei, beispielsweise bezeichnet als »Projektbericht«, im Verzeichnis »Peter« des Laufwerkes DH0: zu verweisen, muß

```
DH0:Peter/Projektbericht
```

eingegeben werden. Dabei ist es bei dieser Schreibweise unwichtig, welches Verzeichnis vorher zum aktuellen Verzeichnis bestimmt wurde.

Wenn Sie auf ein Diskettenlaufwerk oder irgendein anderes Device verweisen, müssen Sie stets den Doppelpunkt eingeben, wie zum Beispiel DF1:.

• Name des Geräts:
• Name des Verzeichnisses/
• eventuell Name des Unterverzeichnisses/
• Name der Datei

Tabelle 2.1 Aufbau einer Dateibezeichnung

Statt des Device-Namens dürfen Sie auch den Diskettennamen zum Aufrufen einer bestimmten Diskette verwenden. Für den Fall, daß sich die gesuchte Datei auf einer Diskette mit dem Namen »Harribald« befindet, wird diese Datei durch Eingabe des vollständigen Namens

```
Harribald:Peter/Projektbericht
```

bezeichnet.

Jede Diskette sollte mit einem möglichst einmaligen Namen bezeichnet werden. Mit dem Diskettennamen sprechen Sie eine Diskette an, unabhängig davon, in welchem Laufwerk sie sich befindet, unabhängig sogar davon, ob sie sich überhaupt in einem Laufwerk befindet (falls die Diskette in keinem der angeschlossenen Laufwerke einliegt, bringt Amiga-DOS einen Requester auf den Bildschirm, der das Einlegen der entsprechenden Diskette verlangt).

Den Diskettennamen geben Sie Ihrer Diskette beim Formatieren. (Weitere Einzelheiten dazu finden sich unter der Anweisung FORMAT in Kapitel 3 dieses Handbuches).

Ein Device-Name ist im Gegensatz zum Diskettennamen kein konstanter Bestandteil der benötigten Bezeichnung. Haben Sie beispielsweise eine Diskette im Laufwerk DF0: erstellt, so kann sie vom Amiga-DOS auch in einem anderen Laufwerk, zum Beispiel DF1:, gelesen werden. Vorausgesetzt, die beiden Laufwerke besitzen das gleiche Diskettenformat (je 3,5-Zoll- oder 5,25-Zoll-Format). Wurde eine Datei »Bernd« auf einer Diskette in Laufwerk DF0: erzeugt, lautet ihre Bezeichnung »DF0:Bernd«. Wird dann die Diskette mit der Datei Bernd in Laufwerk DF1: eingelegt, kann das Amiga-DOS diese Datei mit »DF1:Bernd« lesen.

2.3.5 Vereinbarungen logischer Geräte

Zusätzlich zu den vorher erwähnten physikalischen Geräten (Devices) bietet Amiga-DOS eine Reihe nützlicher logischer Geräte an. Das DOS benutzt diese Devices, um Dateien zu finden, die Programme von Zeit zu Zeit brauchen. Auf diese Weise kann ein Programm auf ein logisches Gerät weisen, unabhängig davon, wie der tatsächliche Zugriffpfad zu dem Verzeichnis lautet, auf das das logische Gerät zeigt. Ein logisches Device kann neu auf ein bestimmtes Verzeichnis gerichtet (engl.: assign) werden. Die logischen Devices werden in diesem Abschnitt wie folgt bezeichnet:

Name	Beschreibung	Verzeichnis
SYS:	Wurzelverzeichnis der Systemdiskette	
C:	DOS-Befehle	C
DEVS:	Diskresidente Gerätetreiber	DEVS
FONTS:	Ladbare Zeichensätze	FONTS
L:	Diskresidente Handler	L
LIBS:	Diskresidente Bibliotheken	LIBS
S:	Scriptdateien	S
T:	Platz für temporäre Dateien	T
ENV:	Umgebungsbereichvariablen	RAM:env
CLIPS:	Wird für das Clipbord-Device benötigt	RAM:clipboards

Tabelle 2.2: Logische Geräte, die das Betriebssystem benutzt

SYS: steht für das Wurzelverzeichnis der SYStem-Diskette oder -Festplatte. Wenn der Amiga gebootet wird, wird SYS: dem Wurzelverzeichnis der Boot-Diskette oder des Festplattenlaufwerks durch das DOS zugewiesen. Für den Fall, daß die Diskette in DF0: den Diskettennamen Start.Disk: hat, weist Amiga-DOS SYS: an Start.Disk: zu. Bootet das System von DH0: und hat DH0: den Namen FAST:, so wird SYS: an FAST: zugewiesen. Nach dieser Zuweisung benutzt jedes Programm, das auf SYS: verweist, das Wurzelverzeichnis der Boot-Diskette.

C: steht für das Befehlsverzeichnis. Wenn Sie eine Anweisung eingeben, zum Beispiel DIR, sucht Amiga-DOS nach der Anweisung zuerst im aktuellen Verzeichnis. Wenn das Betriebssystem den Befehl im aktuellen Verzeichnis nicht findet, sucht es nach C:DIR. Wurde C: einem anderen Verzeichnis zugeteilt, zum Beispiel RAM:c, dann sucht Amiga-DOS Befehle auf der RAM-Disk im Verzeichnis C.

DEVS: Open Device-Funktionsaufrufe suchen hier nach dem Gerätetreiber, falls er nicht bereits im Speicher steht.

FONTS: Open Font-Funktionsaufrufe suchen hier nach ladbaren Zeichensätzen, falls sie nicht bereits in den Speicher geladen wurden.

L: steht für das Handler-Verzeichnis in dem Teile des Dateisystems aufbewahrt werden. Bibliotheken, wie Ram-Handler, Port-Handler und Disk-Validator werden hier aufbewahrt. Amiga-DOS braucht dieses Verzeichnis und die enthaltenen Dateien unbedingt!

LIBS: Open Library-Funktionsaufrufe suchen hier nach der gesuchten Bibliothek, falls sie nicht bereits in den Speicher geladen wurde.

S: steht für das Script-Verzeichnis. Script-Dateien (Batchdateien) enthalten Anweisungssequenzen aus Amiga-DOS-Befehlen, die mit dem Befehl EXECUTE ausgeführt werden können. EXECUTE sucht zuerst nach der Script-Datei im aktuellen Verzeichnis. Wenn EXECUTE sie dort nicht finden kann, sucht es in dem Verzeichnis, auf das mit S: verwiesen wird.

Zusätzlich zu den oben genannten logischen Geräten öffnen viele Programme Dateien in dem Verzeichnis T:. Dieses Verzeichnis wird zum Speichern von Dateien verwendet, die nur zeitweilig benötigt werden. Programme legen in diesem Verzeichnis ihre gegenwärtigen Arbeitsdateien oder Sicherheitskopien der zuletzt bearbeiteten Datei ab. Sollte der Speicherplatz auf der Diskette knapp werden, so ist hier die Suche nach Dateien, die nicht länger benötigt werden, einer der ersten Schritte zur Lösung des Problems. Wenn das Betriebssystem zum ersten Mal geladen wird, weist Amiga-DOS zuerst C: dem Verzeichnis :C zu. Angenommen, Sie laden eine Diskette, die Sie mit dem folgenden Befehl formatiert haben:

```
FORMAT DRIVE DF0: NAME Start.Disk
```

In diesem Fall wird SYS: das Verzeichnis Start.Disk zugeteilt. Das logische Device C: wird auf der gleichen Diskette, als Start.Disk:c, dem Verzeichnis C zugewiesen. Zugleich sind folgende Zuteilungen nach dem Starten des Rechners erledigt:

```
C:      Start.Disk:c
L:      Start.Disk:l
S:      Start.Disk:s
LIBS:   Start.Disk:libs
DEVS:   Start.Disk:devs
FONTS:  Start.Disk:fonts
```

Wenn ein Verzeichnis nicht ansprechbar ist, wird dem Wurzelverzeichnis ein entsprechendes logisches Gerät zugeteilt. Wichtig ist, daß solche Zuweisungen für alle SHELL-Prozesse gelten. Wenn innerhalb eines SHELL-Prozesses eine Zuteilung geändert wird, ändert sie sich auch für alle anderen Prozesse. Soll mit einer selbsterstellten Zeichensatzbibliothek gearbeitet werden, und der Zeichensatz ist auf der Diskette »Zeichensatz Disk« unter »Selbstfonts« gespeichert, muß folgende Anweisung eingegeben werden:

```
ASSIGN FONTS: "Zeichensatz Disk:Selbstfonts"
```

(Dazu Kapitel 3 ASSIGN)

2.3.6 Die RAM-Disks

Amiga-DOS stellt zwei »Laufwerke« zur Verfügung, die im RAM-Speicher des Rechners durch Software simuliert werden.

RAM: ist ein Laufwerk, in das einzelne Dateien hineinkopiert werden können. Die Größe dieser RAM-Disk paßt sich jeweils genau den Platzbedürfnissen der auf ihr befindlichen Dateien an. Der Inhalt von RAM: geht bei einem Reboot verloren.

RAD: ist eine RAM-Disk, die einen fest voreingestellten Speicherbereich belegt, der dem System solange verloren geht, wie RAD: aktiv ist. Die Besonderheit von RAD: ist, daß der Inhalt dieser RAM-Disk nicht bei einem Reboot, sondern nur beim Stromabschalten verlorengeht, und daß Amiga-DOS von RAD: mit installiertem Kickstart1.3 booten kann.

2.4 Theorie und Praxis der Amiga-DOS-Befehle

Eine Anweisung des Amiga-DOS besteht aus dem Namen eines DOS-Befehls und seinen Argumenten, falls welche benötigt werden. Damit eine Anweisung des Amiga-DOS ausgeführt wird, muß der Befehl sofort hinter dem DOS-Prompt eingegeben werden.

Dieser DOS-Prompt besteht aus einer Zahl n und dem Zeichen »>«. Die Variable n steht für die laufende Nummer des jeweiligen SHELL-Prozesses. Diesen Prompt können Sie mit der Anweisung PROMPT, wie in Kapitel 3 beschrieben, ändern.

Die DOS-Befehle benutzen für ihre Ausgabe das Console-Fenster der SHELL, bis ihre Abarbeitung beendet ist. Es können jedoch mehrere Befehle hintereinander eingegeben werden, jeweils nachdem die Eingabezeile des vorherigen Befehls abgeschloßen ist, ohne auf die Beendigung des vorherigen Befehls zu warten. Die Abarbeitung des nächsten Befehls wird aber vertagt, bis der vorherige Befehl seine Ausführung beendet hat. Ach-

tung! Kann ein Befehl aus welchem Grund auch immer nicht ausgeführt werden, so geht Amiga-DOS zur nächsten eingegebenen Anweisung weiter und arbeitet diese ab. Es ist also gefährlich, zu viele Befehle hintereinander einzugeben. Wenn zum Beispiel

```
COPY a TO b
DELETE a
```

eingegeben wurde und der COPY-Befehl nicht ausgeführt werden kann, zum Beispiel, weil die Diskette voll ist, wird DELETE abgearbeitet, und die Datei »a« ist verloren.

2.4.1 Befehle im Hintergrund ausführen lassen

Man kann Amiga-DOS beauftragen, eine oder mehrere Anweisungen im Hintergrund abzuarbeiten. Dazu wird der Befehl RUN verwendet. Er erzeugt einen neuen SHELL-Prozeß niedrigerer Priorität. In diesem Fall werden die nachfolgenden Befehlszeilen simultan zu den mit RUN gestarteten ausgeführt. Zum Beispiel können Sie den Inhalt eines Verzeichnisses zur gleichen Zeit auslesen, zu der eine Textdatei ausgedruckt wird. Dies erreicht man mit:

```
RUN TYPE textdatei TO PRT:
LIST
```

RUN erzeugt einen neuen Hintergrund-SHELL-Prozeß und führt den Ausdruck durch, während im ursprünglichen SHELL-Fenster die Dateien des aktuellen Verzeichnisses aufgelistet werden. Um eine Anweisungszeile mit weiteren Zeilen zu verbinden, wird ein Pluszeichen (+) vor dem Drücken der [Return]-Taste eingegeben. Es können beliebig viele Anweisungszeilen verkettet werden:

```
RUN JOIN textdatei1 textdatei2 AS textdatei +
SORT textdatei TO sortiertext +
TYPE sortierttext TO PRT:
```

2.4.2 Batchprogrammierung mit Script-Dateien

Der Befehl EXECUTE kann dazu verwendet werden, in einer Script-Datei gespeicherte Anweisungen abzuarbeiten, anstatt sie im Direktmodus einzugeben. Der Kommandozeilen-Interpreter liest aus der Datei eine Reihe von Anweisungen, bis er bei der Ausführung eines Befehls auf einen Fehler oder das Ende der Datei stößt. Wird ein Fehler gefunden, so werden nachfolgende Befehle in der Zeile mit RUN oder in der EXECUTE-Datei nicht ausgeführt, es sei denn, der Befehl FAILAT wurde benützt. In Kapitel 3 dieses Handbuchs finden Sie eine ausführliche Beschreibung des Befehls FAILAT. Weitere Eingaben in das SHELL-Fenster, von dem aus der Batch-Prozeß gestartet wurde, sind erst möglich, wenn die Abarbeitung der Script-Datei beendet ist.

2.4.3 Ein- und Ausgabeumleitung von Befehlen

Amiga-DOS ermöglicht die Änderung des Standard-Eingabe- und Ausgabegeräts. Die Symbole »größer als« (>) und »kleiner als« (<) werden als Befehle zur Kennzeichnung einer Umleitung verwendet.

Wird ein Befehl eingegeben, zeigt Amiga-DOS das Ergebnis des Befehls standardmäßig auf dem Bildschirm an. Um die Ausgabe in eine Datei schreiben zu lassen, kann das »>« verwendet werden. Mit dem »<« teilen Sie dem Amiga-DOS mit, daß es die Eingabe aus einer genauer bezeichneten Datei in ein Programm lädt, anstatt sie von der Tastatur zu übernehmen. Die Zeichen »>« und »<« arbeiten wie Verkehrspolizisten, die den Informationsfluß lenken. Um zum Beispiel die Ausgabe von DATE in die Datei »Textdatei« umzuleiten, müssen Sie folgenden Befehl eingeben:

```
DATE > Textdatei
```

In Kapitel 3 dieses Handbuchs ist eine vollständige Erläuterung der Symbole »<« und »>« zu finden.

2.4.4 Unterbrechung eines Programms

Amiga-DOS bietet vier verschiedene Anweisungen, um Abläufe abzubrechen: [Ctrl]+[C], [Ctrl]+[D], [Ctrl]+[E] und [Ctrl]+[F]. Um den momentan arbeitenden Befehl zu beenden, drücken Sie die Kombination [Ctrl]+[C]. In einigen Fällen, z.B. bei EDIT, bewirkt das Drücken von [Ctrl]+[C] nur eine Unterbrechung und ein weiteres Einlesen von EDIT-Befehlen.

Wollen Sie der SHELL mitteilen, daß eine mit EXECUTE initialisierte Befehlsfolge bei Ende des momentan arbeitenden Befehls aufzuhören hat, drücken Sie die Kombination [Ctrl]+[D].

[Ctrl]+[E] und [Ctrl]+[F] werden nur in speziellen Fällen von einigen Befehlen verwendet. Genaueres ist im Amiga-DOS-Programmierer-Handbuch nachzulesen. Beachten Sie bitte, daß diese Abbruch-Flags vom laufenden Programm abgefragt werden müssen und dieses daraufhin seine Arbeit einstellen muß. Amiga-DOS kann kein Programm aus dem Speicher entfernen und die von dem Programm belegten Resourcen freigeben, dies liegt in der Verantwortung des Programmierers.

2.4.5 Das Befehlsformat

Dieser Abschnitt erklärt die von allen Anweisungen des Amiga-DOS verwendete Form und Anordnung der Argumente. Die Zahl und Form der Argumente wird durch Schlüsselwörter bezeichnet. Einige der Schlüsselworte können synonym verwendet werden. In diesem Fall sind sie durch ein Gleichheitszeichen (=) gekennzeichnet. Die folgende Form spezifiziert drei Argumente:

```
ABC, WWW, XYZ = ZZZ
```

Dabei können die Schlüsselworte ZZZ und XYZ äquivalent für dasselbe Argument verwendet werden. Argumente können notwendig oder freiwillig (optional) sein und können auf zwei Arten angegeben werden:

- Durch ihre Lage; die Argumente müssen genau in der angegebenen Reihenfolge stehen.
- Durch Schlüsselwörter; die Reihenfolge spielt keine Rolle, da jedem Argument ein Schlüsselwort vorausgeht.

Wenn zum Beispiel der Befehl KOMMANDO aus einer Datei liest und in eine andere schreibt, heißen die Argumente

```
FROM, TO
```

Der Befehl kann nur durch die Position der Argumente erteilt werden (Schlüsselwörter werden nicht benötigt):

```
KOMMANDO Eingabedatei Ausgabedatei
```

Sie können aber auch Schlüsselwörter verwenden:

```
KOMMANDO FROM Eingabedatei TO Ausgabedatei
```

Mit den Schlüsselwörtern können Sie auch die Reihenfolge der Argumente umkehren:

```
KOMMANDO TO Ausgabedatei FROM Eingabedatei
```

Sie können aber auch beide Formen miteinander kombinieren:

```
KOMMANDO Eingabedatei TO Ausgabedatei
```

Dabei wird das FROM-Argument durch seine Position und das TO-Argument durch sein Schlüsselwort bestimmt. Folgende Form ist jedoch nicht zulässig:

```
KOMMANDO ausgabedatei FROM eingabedatei
```

weil der Befehl voraussetzt, daß Ausgabedatei das erste Argument ist und durch die Lage bestimmt wird (ein Schlüsselwort ist ja nicht vorhanden), also eine FROM-Datei darstellt. Wenn das Argument nicht nur ein einzelnes Wort ist, sondern aus zwei Teilen besteht, die durch Leerzeichen getrennt werden, muß das ganze Argument zwischen zwei Anführungszeichen (") stehen. Wenn das Argument gleich einem Schlüsselwort ist, muß es zur Unterscheidung ebenfalls in Anführungszeichen stehen. Im folgenden Beispiel bezeichnet die Textdatei »Dateiname« das FROM-Argument und der Dateiname »from« das TO-Argument:

```
KOMMANDO "Dateiname" TO "from"
```

Schlüsselworte in den Argument-Listen haben Kennzeichner, die mit den Schlüsselwörtern verbunden sind. Diese Kennzeichner werden durch einen Schrägstrich und einen speziellen Buchstaben angezeigt. Die Bedeutungen der Kennzeichner sind wie folgt:

/A Das Argument wird benötigt und darf nicht weggelassen werden

/K Das Argument muß mit einem Schlüsselwort eingegeben und darf nicht durch seine Position gekennzeichnet werden.

/S Das Schlüsselwort verwendet keine Argumente, da es als Vermittler dient.

Die Kennzeichner /A und /K können kombiniert werden. Die folgende Beschreibung bedeutet zum Beispiel, daß sowohl das Schlüsselwort DRIVE als auch das Argument angegeben werden muß.

```
DRIVE/A/K
```

Betrachten Sie schließlich auch das Format des Befehls TYPE:

```
FROM/A, TO, OPT/K
```

Das erste Argument kann also nach Position oder mit Schlüsselwort angegeben werden. Das zweite muß mit Schlüsselwort eingeleitet werden. Argumente, die nach OPT folgen, sind optional, das heißt, sie können bei Bedarf verwendet werden. Wird OPT jedoch verwendet, muß auch das Schlüsselwort angegeben werden. Zur Verdeutlichung einige Beispiele:

```
TYPE Dateiname
TYPE FROM Dateiname
TYPE Dateiname TO Ausgabedatei
TYPE Dateiname Ausgabedatei
TYPE TO Ausgabedatei FROM Dateiname OPT n
TYPE Dateiname OPT n
TYPE Dateiname OPT n TO Ausgabedatei
```

Dieses Handbuch zeigt bei jedem Befehl alle möglichen Schlüsselworte und Argumente auf. Sollten Sie dennoch einmal an der korrekten Syntax zweifeln, tippen Sie nur den Befehl ein, dahinter ein Leerzeichen und dann ein Fragezeichen. Amiga-DOS zeigt Ihnen dann die korrekte Form. Sie müssen nicht immer im Handbuch nachlesen. Haben Sie nicht die richtige Form gefunden, bricht Amiga-DOS die Ausführung des Befehls mit der simplen Meldung »Bad args« oder »Bad arguments« ab. Der gesamte Befehl muß dann neu eingetippt werden, falls Sie nicht die SHELL unter NEWCON: benutzen. Hier kann durch Tippen auf [Cursor nach oben] der Befehl zurückgeholt und korrigiert werden.

2.5 Der Aktualisierungsprozeß

Haben Sie eine Diskette eingelegt, überprüft Amiga-DOS den ganzen Aufbau der Diskette. Bis der Aktualisierungsprozeß zu Ende ist, können keine Dateien auf dieser Diskette erzeugt werden. Ein Lesen vorhandener Dateien ist jedoch möglich. Ist der Aktualisierungsprozeß erledigt, kontrolliert Amiga-DOS, ob das Datum und die aktuelle Uhrzeit für das System gesetzt wurden. Datum und Uhrzeit werden mit dem Befehl DATE eingestellt. Setzen Sie kein Datum oder keine Uhrzeit für das System, so nimmt Amiga-DOS dafür die Daten der zuletzt erstellten Datei auf der eingelegten Diskette. Dies gibt die Gewißheit, daß neuere Dateiversionen als solche zu erkennen sind, auch wenn das aktuelle Datum und die Zeitangabe nicht stimmen sollten.

2.6 Die gebräuchlichsten Befehle: Eine kleine Übersicht

2.6.1 Einführung in einige Amiga-DOS-Anweisungen

Obwohl alle Anweisungen von Amiga-DOS in Kapitel 2 dieses Buches in allen Einzelheiten erklärt werden, wurde hier eine Zusammenstellung aller wichtigen Anweisungen in ihrer zumeist verwendeten Form erstellt. Denn die meisten Anwender werden die vielen zusätzlichen Argumente, die zur vollständigen Syntax eines Befehls gehören, niemals benutzen. Die unten aufgeführten Anweisungen (zusammen mit ihrem Schlüsselwort) bewirken in Amiga-DOS so wichtige Dinge wie:

```
eine Diskette kopieren (DISKKOPY)
eine Diskette formatieren (FORMAT)
eine Diskette bootfähig machen (INSTALL)
eine Diskette umbenennen (RELABEL)
das Inhaltsverzeichnis einer Diskette lesen (DIR)
Informationen über Dateien lesen (LIST)
ein Datei schützen (PROTECT)
Informationen über eine Diskette lesen (INFO)
das aktuelle Verzeichnis ändern (CD)
Datum und Zeit neu setzen (DATE)
das Ergebnis eines Befehls umleiten (>)
eine Textdatei auf dem Bildschirm ausgeben (TYPE)
eine Datei umbenennen (RENAME)
eine Datei löschen (DELETE)
Dateien kopieren mit zwei Laufwerken (COPY)
Dateien kopieren mit einem Laufwerk (COPY)
ein neues Verzeichnis einrichten (MAKEDIR)
Dateien auf einer Diskette finden (DIR ALL)
logische Devices erzeugen, ändern oder löschen (ASSIGN)
ein neues SHELL-Fenster öffnen (NEWSHELL)
ein SHELL-Fenster schließen (ENDCLI)
```

Wir empfehlen Ihnen, diesen Teil des Buches Schritt für Schritt durchzuarbeiten und jedes einzelne Beispiel nachzuvollziehen. Alle Beispiele sind auf Lauffähigkeit überprüft und leicht überschaubar. Wenn Sie mit dem System dann mehr vertraut sind, können Ihnen die Beispiele als Gedächtnisstütze dienen.

2.6.2 Zu Beginn

Bevor Sie beginnen, sollten Sie Ihre Workbench-Diskette und zwei neue, unbeschriebene Disketten bereitlegen. Bitte bringen Sie nun den Schreibschutz an Ihrer Systemdiskette an!

Die meisten unten beschriebenen Anweisungen gelten sowohl für Systeme mit einem als auch für Systeme mit zwei Laufwerken. Unterscheidet sich die Syntax, wurden beide Ver-

sionen abgedruckt. Nach Eingabe des jeweiligen Befehls drücken Sie bitte die [Return]-Taste. Damit übernimmt der Computer die Ausführung des Befehls. Alle Schlüsselworte und Argumente sind in Großbuchstaben gedruckt. Dies dient nur der besseren Überschaubarkeit der Beispiele. Amiga-DOS erkennt sie auch, wenn die gesamte Zeile mit kleinen Buchstaben getippt wird.

Der Name »df0:« oder erstes Laufwerk bezieht sich in den Beispielen auf das eingebaute Diskettenlaufwerk des Amiga 1000 und des Amiga 500 oder das rechte Laufwerk des Amiga 2000, mit »df1:« oder zweites Laufwerk ist das erste externe 3 1/2"-Laufwerk des Amiga 1000 und des Amiga 500 oder das linke Laufwerk des Amiga 2000 bezeichnet. Ein Semikolon (;) in einer Anweisungszeile grenzt den Kommentar dahinter von dem eigentlichen Befehl ab. Alles, was hinter dem (;) steht, wird von Amiga-DOS ignoriert. Diese Kommentare dienen nur dem Verständnis und brauchen nicht abgetippt zu werden.

2.6.3 Eine Diskette kopieren

Mit diesem Befehl werden ganze Disketten kopiert.

Mit einem Diskettenlaufwerk:

```
DISKCOPY FROM df0: TO df0:
```

mit zwei Laufwerken:

```
DISKCOPY FROM df0: TO df1:
```

Folgen Sie nun den auf dem Bildschirm erscheinenden Anweisungen: Kopieren mit einem Laufwerk: Zuerst schieben Sie die schreibgeschützte Ursprungsdiskette (FROM- oder SOURCE-Disk) in das Laufwerk, dann drücken Sie [Return]. Der Amiga liest nun einen Teil des Disketteninhalts in seinen Speicher. Dann fordert Sie der Rechner auf, die Zieldisk (TO- oder DESTINATION-Disk) einzulegen und wieder [Return] zu drücken. Nun schreibt der Rechner den Inhalt seines Speichers auf die neue Diskette. Diese beiden Vorgänge wiederholen sich nun einige Male, bis der gesamte Disketteninhalt übertragen ist. Kopieren mit zwei Laufwerken: Schieben Sie die schreibgeschützte Ursprungsdiskette (FROM- oder SOURCE-Disk) in das interne Laufwerk und die Zieldisk (TO- oder DESTINATION-Disk) in das externe Laufwerk. Dann drücken Sie die [Return]-Taste. Alles Weitere können Sie getrost dem Computer überlassen. Kopieren Sie nun bitte Ihre Workbench-Diskette und verwahren Sie das Original sicher. Schieben Sie nun die Kopie der Startdiskette in das interne Laufwerk und starten Sie das System von neuem. Falls sie den magischen Drei-Finger-Griff zum Reboot des Amigas noch nicht kennen: Drücken Sie gleichzeitig [Ctrl]+[linke Amiga]+[rechte Amiga].

2.6.4 Eine Diskette formatieren

Vergewissern Sie sich bitte, daß Ihre Workbench-Diskette in Laufwerk dfO: eingelegt ist und Sie eine neue Diskette zur Verfügung haben. Mit dem Befehl FORMAT bereiten Sie eine neue Diskette zur Aufnahme von Daten oder Programmen vor. (Formatieren Sie nicht die eben kopierte Workbench-Diskette! Das Formatieren löscht jegliche Information auf der Diskette ohne Gnade! Nach dem Formatieren befindet sich jede Diskette in einem

LEEREN Zustand). Auf diese Diskette können später einzelne Dateien kopiert werden. Tippen Sie nun:

```
FORMAT DRIVE df0: NAME Ein.Name
```

Folgen Sie nun wieder den Anweisungen auf dem Bildschirm. Legen Sie die neue Diskette, auf der überhaupt keine oder unwichtige Daten sind, in das interne Laufwerk ein und drükken Sie dann Return. Der Amiga kann in allen angeschlossenen Laufwerken Disketten formatieren. Dazu wird nur der jeweilige Device-Name eingesetzt (df0: – df3:). Ist der Formatiervorgang abgeschlossen, geht die Lampe am Laufwerk aus. Entnehmen Sie nun bitte die neu formatierte Diskette. Legen Sie nun wieder Ihre Workbench-Diskette in das Laufwerk df0:.

2.6.5 Eine Diskette bootfähig machen

Für diesen Versuch sollten Sie die Kopie der Workbench-Diskette und die neu formatierte Diskette bereitlegen. Es gibt mehrere Wege, eine startfähige Diskette zu erzeugen, zwei davon stellen wir Ihnen hier vor. Als Startdiskette bezeichnet man eine Diskette, die eingelegt werden kann, wenn der Amiga eine Workbench-Diskette verlangt. Eine formatierte Diskette wird zur bootfähigen Diskette, wenn Sie tippen:

```
INSTALL ?
```

Wenn Sie nur ein Laufwerk benutzen, müssen Sie das Fragezeichen eintippen. Amiga-DOS versucht sonst sofort, die Diskette in df0: zu installieren, und das ist Ihre Workbench-Diskette!

Amiga-DOS antwortet nun:

```
DRIVE/A:
```

entnehmen Sie nun Ihre Workbench-Diskette und legen Sie die neu formatierte ein. Dann tippen Sie:

```
df0:
```

Amiga-DOS kopiert nun die sogenannten Boot-Sektoren auf die Diskette. Der Bootsektor ist nur ein Hinweis für das Betriebssystem, daß der Amiga von dieser Diskette booten kann, der Anwender muß selbst dafür sorgen, daß für das Arbeiten sinnvolle und notwendige Dateien auf der Diskette vorhanden sind (siehe Kapitel 4, Batchdateien, Startup-Sequence).

2.6.6 Umbenennen einer Diskette

Ihre Workbench-Diskette sollte nun wieder in Laufwerk df0: eingelegt sein. Jeder Diskette mit Amiga-DOS-Format kann jederzeit ein neuer Name zugeteilt werden. Dazu dient der Befehl RELABEL. Tippen Sie nun:

```
RELABEL Ein.Name: Ein.neuer.Name
```

In diesem Beispiel wurde der vorhin formatierten Diskette ein neuer Namen zugewiesen.

2.6.7 Das Inhaltsverzeichnis einer Diskette lesen

Ihre Workbench-Diskette sollte nun wieder in Laufwerk df0: eingelegt sein. Mit diesem Befehl lesen Sie das Inhaltsverzeichnis der eingelegten Diskette:

```
DIR oder DIR df0:
```

Diese Form listet den Inhalt des aktuellen Verzeichnisses. Den Inhalt eines anderen Verzeichnisses können Sie lesen, wenn Sie den Pfadnamen des Verzeichnisses eingeben. Für das Verzeichnis C also:

```
DIR df0:c oder DIR c
```

Anstelle der Laufwerksbezeichnung kann auch der Diskettenname verwendet werden. Der Inhalt der eben formatierten leeren Diskette wird also mit:

```
DIR Ein.neuer.Name:
```

auf den Bildschirm gebracht. Heben Sie sich das aber bis später auf, die Disk ist jetzt noch leer.

Manchmal möchte man nicht nur den Inhalt eines Verzeichnisses, sondern den Inhalt der ganzen Diskette durchsehen. Diesem Zweck dient der Befehl DIR mit einer zusätzlichen Option. Geben Sie den folgenden Befehl ein:

```
DIR ALL
```

und alle Verzeichnisse mit allen Unterverzeichnissen und all deren Files erscheinen am Bildschirm. (Mit der [Leertaste] wird die Ausgabe angehalten, mit [Backspace] fortgesetzt!) Um sich einen genaueren Überblick zu verschaffen, leitet man die Ausgabe in ein File um:

```
DIR >Verzeichnisdatei ALL
```

Wichtig ist, daß die Optionen nach dem Dateinamen eingegeben werden. Nun können Sie das File mit TYPE schnell auf den Bildschirm holen, oder mit dem Texteditor ED die Datei lesen und bearbeiten:

```
ED Verzeichnisdatei
```

Mit den Cursor-Tasten können Sie nun vor- und zurückblättern. Mit der Tastenkombination [Esc]+[T]+[Return] geht's an den Anfang. Diese Tastenkombination wird auch [Esc]+[T] genannt. Mit der Tastenkombination [Esc]+[B]+[Return] geht's zum Ende der Datei. Mit der Tastenkombination [Esc]+[M], einer Zahl »n« und [Return] geht's zur Zeile mit der Nummer »n«. Mit der Tastenkombination [Esc]+[Q]+[Return] wird die Arbeit beendet, ohne die Datei auf Diskette zurückzuschreiben. Haben Sie an der Datei etwas verändert, verlangt Amiga-DOS die Eingabe von Y, bevor es den Editor verläßt. Und mit der Tastenkombination [Esc]+[X]+[Return] werden die Änderungen gespeichert.

In Kapitel 7 dieses Handbuches wird ED aber noch genauer beschrieben.

2.6.8 Benutzung des LIST-Befehls

Ihre Workbench-Diskette sollte nun wieder in Laufwerk DF0: eingelegt sein. Mit dem Befehl DIR werden die Namen der Dateien im aktuellen Verzeichnis ausgelesen. Der Befehl LIST macht nun zusätzliche Informationen zugänglich. Tippen Sie ein:

```
LIST oder LIST df0:
```

Amiga-DOS gibt daraufhin Informationen über alle Dateien in dem jeweilige Verzeichnis aus. So zum Beispiel Informationen über die Größe der Datei, über deren Schutzflags (löschbar oder nicht) und über das Datum und die Uhrzeit, zu der es erstellt wurde. Geben Sie zum Befehl LIST und dahinter den Namen eines Verzeichnisses an, werden alle Dateien dieses Verzeichnisses angezeigt.

```
LIST c
```

zeigt alle Dateien des Anweisungsverzeichnisses von Amiga-DOS. Mit den Buchstaben rwedspa wird der Status einer Datei gezeigt.

Dabei steht

- »r« für lesen (read),
- »w« für schreiben (write),
- »e« für ausführbar (execute),
- »d« für löschbar (delete),
- »s« für eine Script-Datei, die auch ohne EXECUTE ausführbar ist (script),
- »p« (pure) für Programme, die reentrant (wiedereintrittsfähig) sind und mit dem Befehl RESIDENT resident gemacht werden können, das heißt diese Programme werden einmal in den Speicher geladen, und können beliebig oft (gleichzeitig) benutzt werden. Das erspart die Ladezeit von Diskette.
- Zuletzt gibt es das Schutzflag »a« für archiviert; mit einem Festplattenbackup-Programm wurde eine Sicherheitskopie der Datei erstellt (archived).

Amiga-DOS beachtet diese Flags. Erscheint zum Beispiel das Flag »d« bei einer Datei nicht in der Ausgabe von LIST, kann diese mit dem Befehl DELETE nicht gelöscht werden.

2.6.9 Schützen einer Datei

Der Befehl PROTECT schützt ein File vor dem versehentlichen Löschen. Auch die Rücknahme dieses Schutzes übernimmt PROTECT. Tippen Sie:

```
DATE > Testfile
PROTECT Testfile
LIST Testfile
```

Nun sind alle Schutzflags gelöscht. Wenn Sie nun versuchen, dieses File mit dem folgenden Befehl zu löschen:

```
DELETE Testfile
```

antwortet Amiga-DOS:

```
Not Deleted - file is protected from deletion
```

Sinngemäß bedeutet das etwa: Nicht gelöscht – File ist vor dem Löschen geschützt. Um nun unser File wieder löschen zu können, muß der Schutz mit der folgende Anweisung entfernt werden:

```
PROTECT Testfile d oder PROTECT Testfile rwed
```

2.6.10 Informationen über eine Diskette lesen

Tippen Sie nun folgenden Befehl ein:

```
INFO
```

Auf dem Bildschirm erscheint alles Wissenswerte über die Workbench-Diskette. Neben dem Diskettennamen wird auch der freie und schon belegte Speicherplatz angezeigt. Benutzen Sie nur ein Laufwerk, erfahren Sie mehr über eine momentan nicht eingelegte Diskette, wenn Sie

```
INFO ?
```

tippen. Amiga-DOS antwortet

```
none:
```

Amiga-DOS hat nämlich auf Ihren Befehl hin den Befehl von der Workbench-Diskette geladen und zeigt die benötigten Argumente und das sind keine (auf englisch none). Entfernen Sie nun die Workbench-Diskette aus dem Laufwerk und legen Sie die gewünschte ein. Drücken Sie nun noch die Return-Taste, zeigt Amiga-DOS alle Informationen über die eben eingelegte Diskette.

2.6.11 Wechsel des aktuellen Verzeichnisses

Bis jetzt haben wir uns nur in der obersten Ebene des hierarchischen Filesystems von Amiga-DOS bewegt. Mehr Informationen über Verzeichnis-Strukturen finden Sie in Kapitel 2.3 dieses Handbuches. Mit dem Befehl:

```
CD
```

erfahren Sie, welches momentan Ihr aktuelles Verzeichnis ist. Haben Sie die Workbench-Diskette im internen Laufwerk und geben dann den Befehl DIR ein, sehen Sie unter anderem auch den Eintrag:

```
c (dir)
```

Dieses Verzeichnis soll nun das aktuelle werden. Geben Sie dazu ein:

```
CD c oder CD df0:c
```

Lesen Sie nun nochmals das Verzeichnis, dann werden Sie feststellen, daß nun der Inhalt des Verzeichnisses C aufgelistet wird. In die oberste Dateiebene einer Diskette kommen Sie wieder, wenn Sie dem Befehl einen Doppelpunkt folgen lassen:

```
CD : oder CD df0:
```

2.6.12 Datum und Zeit neu stellen

Die Amiga-DOS-Uhr wird mit dem Befehl DATE gestellt:

```
DATE 12:00:00 12-jan-87
```

Die Systemuhr enthält jetzt diese Zeit.

2.6.13 Umleiten der Ausgabe in eine Datei

Normalerweise wird jedes von einem Befehl gelieferte Ergebnis auf dem Monitor angezeigt. Sie können diese Ausgabe jedoch mit dem Zeichen »>« statt dessen in eine Datei schreiben. Hier ein Beispiel:

```
DATE >Datenfile
```

Hat der Rechner diesen Befehl ausgeführt, finden Sie im aktuellen Verzeichnis einen Eintrag »Datenfile«. Existierte diese Datei bereits vorher, wurde der alte Inhalt überschrieben! Mit der Syntax:

```
DATE >Ein.neuer.Name:Datenfile
```

wird die Datei auf die vorhin formatierte Diskette geschrieben. Dazu werden Sie von Amiga-DOS aufgefordert, diese Diskette in das Laufwerk einzulegen. Warten Sie dann das Erlöschen der LED am Laufwerk ab und legen Sie die Workbench-Diskette wieder ein. Tippen Sie nun:

```
DIR Ein.neuer.Name:
```

Wieder werden Sie zum Diskettenwechsel aufgefordert. Nun finden Sie auf der neuen Diskette ebenfalls eine Datei mit Namen »Datenfile«.

2.6.14 Eine Textdatei auf dem Bildschirm ausgeben

Der Inhalt einer Textdatei kann mit dem Befehl TYPE lesbar gemacht werden:

```
TYPE Daten
```

Wollen Sie die Ausgabe vorübergehend anhalten, drücken Sie einfach auf die Leertaste. Weiter geht sie dann mit [Backspace]. Soll die Ausgabe vor dem Dateiende abgebrochen werden, halten Sie bitte die [Ctrl]-Taste gedrückt und tippen ein »c«. Um eine Testdatei auf einer nicht aktuellen Diskette zu lesen, tippen Sie:

```
TYPE Ein.neuer.Name:Datenfile
```

2.6.15 Umbenennen einer Datei

Der Name einer Datei kann mit dem Befehl

```
RENAME FROM Datenfile TO neuer.Name
```

oder einfach mit

```
RENAME Datenfile neuer.Name
```

geändert werden.

Beachten Sie bitte, daß die meisten Amiga-DOS-Anweisungen in dieser abgekürzten Form verwendet werden dürfen. Zu Beginn Ihrer Arbeit mit Amiga-DOS wird aber die lange Variante mit Schlüsselwörtern empfohlen, da deren Funktion leichter zu durchschauen ist. In der Kurzübersicht sind alle alternativen Formen aufgezeigt. Überprüfen Sie nun mit TYPE, daß sich der Inhalt der Datei nicht geändert hat:

```
TYPE neuer.Name
```

2.6.16 Eine Datei löschen

Um wieder Platz auf Ihren Amiga-DOS-Disketten zu schaffen, werden Sie von Zeit zu Zeit nicht mehr genutzte Dateien oder Programme löschen wollen. Dies ermöglicht der Befehl DELETE.

Achtung! Eine einmal gelöschte Datei kann nicht wieder herbeigeholt werden! Vergewissern Sie sich, ob die Datei wirklich nutzlos ist. Hier nun eine kleine Beispielsequenz. Sie erstellt ein kleines Textfile, gibt dieses File auf dem Bildschirm aus, löscht es und versucht es dann erneut zu lesen:

```
DIR      >Verzeichnisfile
TYPE     Verzeichnisfile
DELETE   Verzeichnisfile
TYPE     Verzeichnisfile
```

Nach dem letzten Befehl sehen Sie auf dem Bildschirm:

```
»Can't open Verzeichnisfile«
```

Das heißt auf deutsch etwa: ich kann Verzeichnisfile nicht öffnen. Fragen Sie danach warum (englisch why):

```
WHY
```

Dann antwortet Amiga-DOS darauf mit:

```
Last command failt because object not found
```

Der letzte Befehl kann nicht ausgeführt werden, weil es das Objekt nicht gibt.

2.6.17 Kopieren von Dateien mit zwei Laufwerken

Mit zwei Laufwerken ist das Kopieren von Dateien ein Kinderspiel:

```
COPY FROM df0:Ursprungsdatei TO df1:Zieldatei
```

oder

```
COPY df0:Ursprungsdatei df1:Zieldatei
```

Selbstverständlich können Sie auch eine Datei von DF1: nach Laufwerk DF0: kopieren.

2.6.18 Dateien mit einem Laufwerk kopieren

Mit einem Laufwerk Dateien zu kopieren, erfordert etwas mehr Arbeit. Zuerst sollten einige DOS-Befehle resident gemacht werden, so daß sie zum Arbeiten nicht mehr dauernd von Diskette nachgeladen werden müssen:

```
RESIDENT DF0:C/CD
RESIDENT DF0:C/COPY
```

Legen Sie dann die Ursprungsdiskette in das Laufwerk. (Für dieses Beispiel kopieren wir das Werkzeugprogramm EXECUTE aus dem Verzeichnis C von der Workbench-Diskette, die bereits im Laufwerk ist.) Tippen Sie:

```
COPY df0:c/execute RAM:execute
```

oder

```
COPY df0:c/execute RAM:
```

Entnehmen Sie nun die Ursprungsdiskette dem Laufwerk und legen Sie die Zieldiskette ein. Tippen Sie:

```
COPY RAM:execute df0:execute
```

oder

```
COPY execute df0:execute
```

Entnehmen Sie nun die Zieldiskette wieder und schieben Sie Ihre Workbench-Diskette in das Laufwerk. Tippen Sie nun:

```
CD df0:
```

und Sie befinden sich wieder auf der Workbench-Diskette.

Mit dem folgenden Befehl wird auch der Speicherplatz, den die RAM-Disk belegte, wieder frei:

```
DELETE RAM:execute
```

(Allerdings bleibt dabei 1 Kbyte für das Verzeichnis der RAM-Disk zurück. Ein Verlust, der zu verschmerzen ist.)

2.6.19 Ein neues Verzeichnis einrichten

Ein neues Verzeichnis (eine neue Schublade) im aktuellen Verzeichnis wird mit dem Befehl

```
MAKEDIR neues_Verzeichnis
```

eingerichtet.

Um eine Datei aus einem Verzeichnis in ein anderes auf der gleichen Diskette zu befördern, benutzen Sie den Befehl

```
RENAME FROM dateiname TO neuVerzeichnis/dateiname
```

Mit dem Befehl

```
DIR neuVerzeichnis
```

können Sie sich von der Funktion überzeugen.

2.6.20 Anweisungen beim Systemstart automatisch durchführen lassen

Im Verzeichnis S der Workbench-Diskette befindet sich eine Datei namens »Startup-Sequence«. Dies ist eine sogenannte Script-Datei, also eine Ansammlung von CLI-Anweisungen, die automatisch ausgeführt werden, wenn das System gestartet wird.

Die beiden letzten Anweisungen dieser Datei sind LOADWB (lädt Workbench) und ENDCLI (beendet den SHELL-Prozeß und übergibt die Kontrolle des Systems an die Workbench). Mit dem Programm MEMACS von der Extras-Diskette können Sie nun eine eigene Startup-Sequence erzeugen. Später wird genauer auf den Befehl EXECUTE eingegangen. Durch eine geänderte Startup-Sequence können Programme sofort beim Systemstart geladen und gestartet werden. So können Sie automatisch eine SHELL starten, wenn Sie vor den Befehl ENDCLI den Befehl NEWSHELL einsetzen. Im Kapitel 4 »Script-Dateien« wird das Thema eigene Startup-Sequence genauer unter die Lupe genommen.

2.6.21 Logische Geräte erzeugen, ändern oder löschen

Manchmal möchten Sie mit einer anderen als der Startdiskette weiterarbeiten. Haben Sie zum Beispiel den Rechner mit der Workbench-Diskette gestartet und wollen mit der Diskette »Ein neuer.Name« weiterarbeiten, geht das mit einem Laufwerk nur mit vertretbarem Aufwand, wenn auf der Diskette Ein.neuer.Name das c-Verzeichnis mit den Files für die Befehle des CLI zu finden ist. Um dann auf dieses File zugreifen zu können, verwenden Sie den Befehl ASSIGN. Verwenden Sie diesen Befehl nicht, sieht die weitere Arbeit so aus:

```
CD Ein.neuer.Name:
```

Amiga-DOS antwortet »Insert Ein.neuer.Name into any drive«. Legen Sie also die Diskette »Ein.neuer.Name« in das Laufwerk, und tippen Sie dann:

```
DIR
```

Amiga-DOS antwortet nun:

```
insert »Name Ihrer Startdiskette« into any drive.
```

Da Amiga-DOS seit dem Start weiß, daß alle DOS-Anweisungen im Verzeichnis »c« auf der Startdiskette stehen, will es jetzt diese, und nur diese, Diskette haben. Amiga-DOS liest also dann den Befehl ein und fordert Sie auf, die aktuelle Diskette einzulegen, also »Ein.neuer.Name«. Bei jedem weiteren DOS-Befehl sind nun ebenfalls zwei Diskettenwechsel nötig. Um dies zu vermeiden, tippen Sie folgendes:

```
ASSIGN c: Ein.neuer.Name:c
```

Amiga-DOS fordert:

```
insert Ein.neuer.Name into any drive.
```

Von nun an liest Amiga-DOS seine Anweisungen aus dem Verzeichnis c der Diskette Ein.neuer.Name. Nun brauchen Sie nur noch den folgenden Befehl einzutippen und Sie sind am Ziel:

```
CD Ein.neuer.Name:
```

Es gibt noch mehr sinnvolle Zuweisungen unter Amiga-DOS. Wenn Sie folgenden Befehl eingeben, werden diese aufgelistet:

```
ASSIGN LIST
```

Sprechen Sie zum Beispiel aus einem Programm heraus ein am seriellen oder parallelen Port angeschlossenes Gerät (Drucker oder Modem) an, sucht Amiga-DOS im Verzeichnis DEVS: nach den zur Ansprache dieser Geräte nötigen Anweisungen. Haben Sie alle System-Verzeichnisse auf die neue Diskette kopiert, brauchen Sie die Startdiskette nicht mehr. Der Inhalt einer Script-Datei für die Diskette »Ein.neuer.Name« sieht dann so aus:

```
ASSIGN SYS: Ein.neuer.Name:
ASSIGN S: Ein.neuer.Name:s
ASSIGN DEVS: Ein.neuer.Name:devs
ASSIGN L: Ein.neuer.Name:l
ASSIGN FONTS: Ein.neuer.Name:fonts
ASSIGN LIBS: Ein.neuer.Name:libs
```

Um diese Script-Datei zu erstellen, tippen Sie:

```
COPY FROM * TO umschaltfile
```

Dann geben Sie jede einzelne Zeile ein. Nach der letzten Zeile drücken Sie schließlich [Ctrl]+[/]. Das Sternchen (*) steht für das aktuelle Fenster und die Tastatur. Auf diese Weise können kleinere Files ohne MEMACS erstellt werden.

2.6.22 Ein neues SHELL-Fenster öffnen

Amiga-DOS ist ein multitaskingfähiges System. Sie können also zu gleicher Zeit in mehreren Fenstern verschiedene Aufgaben erledigen lassen. Jede Aufgabe braucht jedoch ein eigenes Fenster. Eine neue SHELL erscheint mit dem Befehl:

```
NEWSHELL
```

Ein CLI (die alte unkomfortable SHELL) erscheint mit:

```
NEWCLI
```

Das Prompt, das in diesem Fenster erscheint, enthält vor dem »größer als«-Zeichen (>) die laufende Nummer des SHELL-Fensters und den mit CD gesetzten aktuellen Pfad:

```
1.Harribald/Text>
```

Geben Sie nun ein:

```
NEWSHELL
```

Daraufhin erscheint ein neues Fenster mit dem folgenden Prompt

```
2.Harribald>
```

Dieses Fenster kann nun größer oder kleiner gemacht und über den ganzen Bildschirm bewegt werden.Wollen Sie dem zweiten Task Anweisungen erteilen, klicken Sie in das Fenster und tippen den Befehl ein. Versuchen Sie folgendes:

1. Klicken Sie in Fenster 1> und tippen Sie:

```
DIR df0:c
```

2. Klicken Sie schnell in Fenster 2> und dann:

```
INFO
```

Beide SHELL-Fenster erfüllen ihre Aufgaben zur gleichen Zeit. Sie sind dabei nicht auf diese zwei Fenster beschränkt. Steht genug Speicherplatz zur Verfügung, können 20 oder mehr Fenster gleichzeitig geöffnet sein.

2.6.23 Ein SHELL-Fenster schließen

Der jeweilige SHELL-Prozeß wird beendet, indem Sie mit dem Auswahl-Knopf der Maus in das Fenster klicken und dann den Befehl

```
ENDCLI
```

eingeben. Das war's schon; das Fenster verschwindet daraufhin vom Bildschirm.

2.6.24 Zum Schluß dieses Abschnittes

Diese Beispiele sollten Ihnen nur eine kleinen Einführung in die Fähigkeiten der AmigaDOS SHELL geben. Sehr viele der Anweisungen sind in diesem Teil des Buches nicht angeschnitten worden, die Syntax anderer Anweisungen ist stark vereinfacht dargestellt worden und viele Optionen der Anweisungen wurden gar nicht erläutert.

Kapitel 3 enthält die genaue Beschreibung mit allen Optionen zu jedem einzelnen Befehl. Sehen Sie bitte dort nach, wenn etwas an den Beispielen unklar geblieben ist.

3 Die Amiga-DOS-Befehle

3.0 Verwendete Fachausdrücke

In diesem Handbuch finden sich einige Begriffe, die manchem Leser unbekannt sein könnten. Darum eine kleine Erklärung der wichtigsten Fachausdrücke:

- *Booten/Bootvorgang*: Starten des Rechners und Einlesen des Betriebssystems.
- *Default*: Eine Grundeinstellung beziehungsweise die Einstellung, die gültig ist, wenn Sie nichts anderes bestimmen.
- *Device*: Ein Device ist ein Gerät, wie ein Floppylaufwerk oder der Drucker. Damit kann man aber auch eine rechnerinterne Adresse bezeichnen, die wie ein physikalisches Gerät agiert, wie zum Beispiel eine RAM-Disk.
- *Device-Name*: Der Name, der ein Device von einem anderen unterscheidet und es bezeichnet. Device-Namen folgt immer ein Doppelpunkt (:). Beispiele: CON:, DF0:, PRT: und so weiter.
- *File handle*: Ein interner Wert von Amiga-DOS, der eine offene Datei oder ein offenes Device repräsentiert.
- *Logisches Device (Gerät)*: Ein Name, der einem Verzeichnis mit dem Befehl ASSIGN zugeordnet wird und dann wie ein Device verwendet werden kann.
- *Objekt Code*: Binäre Ausgabe eines Assemblers oder Compilers (noch nicht lauffähiger Maschinencode) und binäre Eingabe an einen Linker.
- *Stream*: Eine geöffnete Datei oder ein Device, die mit einem Filehandle-Wert verbunden ist. Zum Beispiel kann der Eingabe-Stream aus einer Datei kommen und der Ausgabe-Stream zum Bildschirm (NEWCON:) gehen.
- *System-Diskette*: Die Diskette, auf der sich die Workbench und die DOS-Befehle befinden.
- *Volume-Name*: Ein Name, der eine Diskette oder Festplatte bezeichnet.

Auf den folgenden Seiten finden Sie Beschreibungen der einzelnen Befehle, die Sie in einem SHELL-Fenster verwenden können.

3.0.1 Nach folgendem Schema werden die Befehle dokumentiert:

Format:	[<anweisung>];[<kommentar>]
Muster:	"Anweisung";"Kommentar"
Zweck:	Einen Kommentar; an einen Befehl anhängen.
Pfad:	Wo der Befehl zu finden ist.

Beschreibung:

Das CLI ignoriert alles in einer Befehlszeile nach dem »;«.

Beispiel(e):

```
;Diese Zeile ist nur ein Kommentar
```

diese Zeile wird vom Rechner nicht beachtet.

```
COPY <file> TO prt: ;druckt das File
```

kopiert die Datei <file> zum Gerät Drucker und ignoriert den Teil: »;druckt das File«

Siehe auch: EXECUTE

BEFEHL	Amiga-DOS-Befehle werden in Großbuchstaben geschrieben, um sie vom restlichen Text deutlich zu unterscheiden.
< >	Spitze Klammern schließen notwendige Argumente ein, die angegeben werden müssen. <Datei> bedeutet z.B., daß der Amiga-DOS-Befehl eine Datei bearbeitet, deren Name angegeben werden muß.
[ ]	Eckige Klammern schließen Optionen ein. Optionale Argumente werden von einigen Anweisungen akzeptiert, sind aber nicht zwingend erforderlich.
{ }	Geschweifte Klammern bedeuten, daß in diesem Befehlsteil beliebig viele Argumente angegeben werden dürfen, deren Angabe jedoch nicht zwingend ist.
. . .	Drei Punkte weisen auf eine Folge hin, die fortgesetzt werden kann.
*	Gleiche Bedeutung wie drei Punkte ...
|	Ein Senkrechtstrich wird verwendet, um eine Auswahl zwischen Optionen in einer Liste zu kennzeichnen, die jeweils gleichwertig benutzt werden können. [OPT A|B|AB] bedeutet z.B., daß die Option A, die Option B oder beide (AB) gewählt werden können (logisches ODER).
<Argument>/A	Das Argument muß angegeben werden, damit der Befehl arbeitet.
<Option>/K	Das Schlüsselwort (K wie Key) der Option muß gegeben werden, auch wenn weitere Parameter angegeben sind. Eine der Options-

gruppen des Befehls DIR ist [OPT A|I|AI|D]. Die Schablone lautet "OPT/K". Das bedeutet, daß das Wort OPT zusammen mit der gewählten Option eingegeben werden muß.

<Option>/S — Die Option arbeitet als Schalter. Der Name der Option muß eingegeben werden, um sie zu wählen. Der Befehl DIR verfügt z.B. auch über eine Option [ALL]. Das Wort ALL muß geschrieben werden, wenn Sie diese Option benutzen wollen.

3.1 Die SHELL

Die SHELL ist neben der Workbench die zweite Benutzerschnittstelle von Amiga-DOS. Die Workbench bietet dem Benutzer viel Komfort aber leider sehr eingeschränkte Möglichkeiten.

Die SHELL ist eine alphanumerische Schnittstelle, unter der der Benutzer dem Computer die Arbeitsabläufe durch Tastatureingaben vermittelt. Die SHELL dient als Editor für die Kommandozeilen und als virtuelles Terminal für alphanumerische Ausgaben. Nachdem die Eingabe der Kommandozeile mit [RETURN] abgeschlossen wurde, übergibt die SHELL die Eingabezeile an den Kommandozeilen-Interpreter. Dieser versucht zuerst im aktuellen Verzeichnis ein Programm des angegebenen Namens zu finden, und es zur Ausführung zu laden. Falls kein Programm gefunden wird, werden alle Befehlsverzeichnisse durchsucht, deren Namen dem System durch den Befehl PATH mitgeteilt worden sind.

Die DOS-Befehle dienen als FILTER, sie lesen in der Regel einen Datenstrom von der Standardeingabe (Tastatur) und schreiben auf die Standardausgabe (SHELL-Fenster), nachdem sie den Datenstrom bearbeitet haben. Bei korrekter Ausführung eines Befehls meldet dieser den Rückgabewerte Null (0) an die SHELL zurück, bei Auftreten eines Fehlers einen positiven Wert (siehe FAILAT).

Beim Starten der SHELL durch den Befehl NEWSHELL oder durch Anklicken des SHELL-Icons auf der Workbench wird eine Script-Datei aufgerufen, in der der Benutzer bestimmte, eigene Befehlssequenzen ausführen lassen kann, um Voreinstellungen vorzunehmen. Diese Datei befindet sich im Ordner S: und heißt »Shell-Startup«.

In der Datei »Shell-Startup« auf der originalen Workbench 1.3 wird mit Hilfe des Befehls PROMPT zum Beispiel die Eingabeaufforderung, der »Prompt«, so eingestellt, daß zusätzlich zur Nummer des SHELL-Prozesses auch das aktuelle Verzeichnis mit ausgegeben wird. Zum Beispiel:

```
1.FAST:prefs
```

Die Datei S:Shell-Startup ist eine reine ASCII-Datei und kann mit Texteditoren wie ED und MicroEmacs bearbeitet werden.

Wenn ein Icon auf der Workbench ausgewählt (einmal angeklickt) wird und danach in der Workbench Menüleiste der Menüpunkt INFO angewählt wird, erscheint ein Fenster, in dem die Informationen über das Programm dargestellt werden, die in dem zum Programm

gehörigen Icon enthalten sind. Darunter sind einige Voreinstellungen für den Programmstart unter dem Punkt TOOL TYPES möglich. Mit ADD und DELETE können hier neue Zeilen eingefügt oder ungewollte gelöscht werden.

Für ein SHELL-Fenster sind folgende TOOL-TYPES möglich:

```
WINDOW= <Parameterliste für Fenster>
```

Die Parameterliste gibt die Größe, die Lage und den Console-Handler an, der für das Fenster verwendet werden soll.

Das Format für eine Fensterfestlegung lautet:

```
"Handler:x/y/w/h/Name", wobei gilt:
```

Handler CON: oder NEWCON:

x Anzahl der Bildpunkte zwischen der linken Bildschirm- und der linken Fensterkante

y Anzahl der Bildpunkte zwischen der oberen Bildschirm- und der oberen Fensterkante

w Breite des Fensters von der linken oberen Ecke aus in Bildpunkten

h Höhe des Fensters von der linken oberen Ecke aus in Bildpunkten

Name Titel des Fensters

Als Handler sollte ab jetzt immer der neue Handler »NEWCON:« verwendet werden, der ein komfortables Editieren der Kommandozeile ermöglicht. NEWCON: ist ein Ersatz für CON: und muß mit dem Befehl MOUNT in der Datei »S:Startup-Sequence« angemeldet werden. (Die originale Workbench 1.3 enthält einen entsprechenden Befehl).

Der Befehl MOUNT verwendet Einträge aus der Datei »DEVS:Mountlist«, die die Gerätetreiber und Handler genauer beschreiben. (Siehe Kapitel 5, Mountlist)

Der Eintrag

```
STACK= <Größe>
```

ist gleichbedeutend mit dem DOS-Befehl Stack und dient zum Festlegen der Größe des Stapelspeichers, der von allen Prozessen verwendet wird, die von dieser SHELL gestartet werden.

Damit die SHELL funktionsfähig ist, muß das Programm SHELL-SEG aus dem L:-Verzeichnis resident in die Systemliste eingetragen werden. Obwohl sich SHELL-SEG zusammen mit den Amiga-Handlern im Verzeichnis L: befindet, ist es kein Handler, sondern ein Ersatz für den Kommandozeilen-Interpreter CLI.

Der Befehl

```
RESIDENT L:SHELL-SEG SYSTEM
```

installiert den SHELL-Server im System, und sollte am besten gleich in der Datei Startup-Sequence eingetragen werden, damit nach jedem Reboot gleich die SHELL anstatt des ein-

fachen CLI verfügbar ist. Mit dem Icon SHELL auf der Workbench-Diskette oder NEWSHELL kann ab dann eine neue interaktive SHELL gestartet werden.

3.1.1 ESCAPE-SEQUENCE und [Ctrl]-CODES

Die SHELL interpretiert folgende Escape-Sequenzen, die zum Beispiel mit ECHO gesendet werden können (»*e[« der Stern (*) leitet den Escape-Character (e[) ein, danach wird der eigentliche Befehl gesendet):

Löschen des Fensters und Reset aller Einstellungen:

```
* e[c
```

Schriftarten:

normal	*e[0m
fett	*e[1m
kursiv (italic)	*e[3m
unterstrichen	*e[4m
invers	*e[7m
unsichtbar	*e[8m

Vordergrundfarbe (bezüglich der Originaleinstellung):

normal	*e[30m
weiß	*e[31m
schwarz	*e[32m oder *e[2m
orange	*e[33m

Hintergrundfarbe (bezüglich der Originaleinstellung):

normal	*e[40m
weiß	*e[41m
schwarz	*e[42m
orange	*e[43m

Spezialkommandos:

Breite des SHELL-Fenster	*e[nu , n ist die Breite in Buchstaben
Höhe des SHELL-Fenster	*e[nt , n ist die Zeilenanzahl
Abstand in Pixels von links	e[nx , n ist die Anzahl in Pixel
Abstand in Pixels von oben	*e[ny , n ist die Anzahl in Pixel

Ctrl-Tastenkombinationen:

[Ctrl]+[C]	
[Ctrl]+[D]	
[Ctrl]+[E]	
[Ctrl]+[F]	Abbruchsignale (siehe auch BREAK)
[Ctrl]+[G]	Aufblitzen des Bildschirms ohne BEEP
[Ctrl]+[H]	wie [Del]
[Ctrl]+[I]	wie [TAB]
[Ctrl]+[J]	wie [Return]

Ctrl+K	Cursor eine Zeile nach oben
Ctrl+L	mit anschließendem Return löscht das Fenster
Ctrl+M	wie Carriage Return + Linefeed
Ctrl+N	schaltet auf alternativen Zeichensatz
Ctrl+O	schaltet auf normalen Zeichensatz

3.1.2 Editieren von Befehlszeilen

Wird die SHELL anstatt mit dem Handler NEWCON: mit CON: betrieben, verliert sie fast alle unten beschriebenen Fähigkeiten, deshalb sollte wenn immer nur möglich NEWCON: verwendet werden.

Mit den Tasten Cursor rechts und Cursor links kann der Cursor über den Text in der Kommandozeile bewegt werden. Shift+Cursor rechts setzt den Cursor an das Ende, Shift+Cursor links an den Anfang der Befehlszeilen. Del löscht das Zeichen unter dem Cursor Backspace das Zeichen links vom Cursor.

Ctrl+K	löscht den gesamten Text von der Cursorposition nach rechts bis zum Zeilenende.
Ctrl+U	löscht den Text von der Cursorposition bis zum Zeilenanfang.
Ctrl+X	löscht die ganze Zeile.
Ctrl+W	läßt den Cursor an die nächste Tabulatorposition springen.

Mit Cursor oben blättert man in den zuletzt eingegebenen Befehlszeilen zurück, kann also eine bestimmte früher eingegebene Befehlszeile zum Bearbeiten und Ausführen zurückholen. Ctrl+R und Shift+Cursor oben holen die erste Befehlszeile zurück, die noch in den Puffer paßt. Dieser Puffer ist 2 Kbyte groß und wird nach dem FIFO-Prinzip verwaltet. Wenn der Puffer gefüllt ist, werden die ältesten Befehle nach und nach gelöscht, um so Platz für neue Eingabezeilen zu schaffen.

Cursor unten blättert wieder vorwärts im Befehlszeilenspeicher, Shift+Cursor unten setzt den Cursor auf das Ende des Befehlspuffer zur Eingabe einer neuen Befehlszeile, falls man sich irgendwo mitten zwischen den alten Befehlszeilen befindet.

3.1.3 Befehls-Makros

Die SHELL erlaubt, daß Befehlszeilen zu Makros zusammengefaßt werden können. Dazu dient das in die SHELL integrierte Befehlswort ALIAS. So können häufig benutzte Befehle abgekürzt oder Befehlsnamen aus anderen Betriebssystemen nachgebildet werden:

```
alias <Abkürzung> <Zeichenfolge>
alias xcopy copy [] clone
alias endshell endcli
alias clear echo "*E[0;0H*E[J"
alias ps status full
alias mv rename
```

In einem Makro gilt »[]« als Platzhalter für die Eingabe eines Parameters durch den Benutzer. Wenn die SHELL eine Eingabe als Makro erkennt, wird der Makroname durch die Zeichenfolge ersetzt, der Rest der mit dem Makronamen eingegebenen Zeichenfolge wird angefügt und danach dem Kommandozeileninterpreter zum Ausführen übergeben.

Die Makros gehören zur privaten Umgebung einer SHELL und werden nicht auf eine neu erzeugte SHELL vererbt. Häufig verwandte Makros sollten deshalb in die Datei »S:Shell-Startup« eingefügt werden.

ALIAS ohne Parameter listet alle zur Zeit verfügbaren Makroersetzungen auf. Ein Makro wird gelöscht, indem der Makroname allein, ohne Ersatztext eingegeben wird.

3.2 Die DOS-Befehle

Ein-/Ausgabe-Umleitung > <

Format: <Anweisung> [>Ausgabefilename] [<Eingabefilename] [<Anweisungsargumente>*]

Muster: "Anweisung">"TO"<"FROM" "Argumente"

Zweck: Umleiten der Ein- und Ausgaberichtung von Daten oder Argumenten.

Beschreibung:

Die Symbole »<« oder »>« ändern die Standard-Ein- beziehungsweise Ausgabe eines Befehls. Die Spitze des Zeichens weist in die Richtung des Datenflusses. Die Ausgabe erfolgt normalerweise in das aktuelle Fenster. Wird jedoch nach dem Befehl (also noch vor eventuellen Argumenten!) ein »größer als« (>) und ein Dateiname angegeben, werden die Ausgabedaten dieses Befehls in eine Datei geschrieben, die unter angegebenen Namen neu angelegt wird. Existiert diese Datei noch nicht, wird sie neu eingerichtet (statt einer Datei kann natürlich jedes Device Verwendung finden, zum Beispiel PRT:).

Erwartet der Befehl dagegen weitere Parameter und wird nach dem Befehl das Symbol »<« und ein Dateiname eingegeben, liest der Rechner alle weiteren Eingaben aus dem angegebenen File. Die Schlüsselworte TO und FROM brauchen nicht eingegeben zu werden. Die Änderung des Datenstromes gilt nur für diesen einen Befehl. Nach der Ausführung sind wieder Tastatur und aktuelles Fenster Ein- und Ausgabe-Devices.

Beispiel(e):

```
DATE >Tagebuchdatei
```

Dieser Befehl schreibt das aktuelle Datum und die Uhrzeit in die Datei Tagebuchdatei.

```
programm <Eingabe
```

Der Befehl Programm übernimmt Daten von der Datei »Eingabe«.

```
LIST >temp
SORT temp TO *
```

Die Liste der Dateien wird in die Datei »temp« geschrieben und dann sortiert auf dem Bildschirm ausgegeben.

```
ECHO >2.datum 02-jan-87
DATE <2.datum ?
DELETE 2.datum
```

Erzeugt die Datei »2.datum« mit dem Inhalt »02-jan-87 <linefeed>«, leitet diese Datei dem Befehl DATE als Argument zu und löscht dann die Datei wieder.

Das Fragezeichen nach dem Dateinamen muß eingegeben werden, da DATE sonst nur Eingaben von der Tastatur annimmt. Die Kombination von Fragezeichen und Eingabeumleitung kann noch interessanter genutzt werden. So ist es denkbar, einige der Parameter beim Aufruf eines Befehls einzugeben, und alle übrigen benötigten Argumente von einer Datei zu lesen.

Mit

```
ECHO >T:HELP_DIR "ALL"
```

wird eine Datei angelegt, in der zwei Optionen für den Befehl DIR abgespeichert werden, um sie mit Hilfe von Eingabeumleitung an den DIR-Befehl übergeben zu können. Folgendes ist denkbar:

```
DIR <T:HELP_DIR DH0:FONTS FILES ?
```

Siehe ECHO und DIR für eine genaue Beschreibung der Befehle.

ADDBUFFERS

Format: ADDBUFFERS <Laufwerk>: <nn>

Muster: ADDBUFFERS "LAUFWERK/A,PUFFER/A"

Zweck: Reduzierung der Disk-Zugriffszeiten durch Vergrößerung des Disk-Pufferspeichers.

Pfad: C:ADDBUFFERS

Beschreibung:

Dieser Befehl fügt <nn> Pufferspeicher zu der bestehenden Liste von Sektor-Zugriffspuffern für <Laufwerk> hinzu. Dadurch kann die effektive Disketten-Zugriffszeit teilweise erheblich reduziert werden. Der einzige Nachteil dieses Befehles: Er verbraucht Speicherplatz. Jeder neu hinzugefügte Arbeitspuffer benötigt 512 Byte RAM.

Fügen Sie mehr als 25–30 zusätzliche Pufferspeicher für ein Gerät hinzu, das das alte Filing System benutzt, hat das kaum noch Auswirkungen auf die Zugriffszeiten, und verschwendet wertvollen RAM-Speicher.

Bei der Benutzung einer Festplatte unter Fast-Filing-System beschleunigt das Hinzufügen von Pufferspeichern immer den Plattenzugriff.

Beispiel(e):

```
ADDBUFFERS DF1: 25
```

Hier werden 25 zusätzliche Puffer für Drive DF1: reserviert. Falls Sie nicht genau wissen, wieviel Pufferspeicher für Ihre Anwendung optimal ist, dann sollten Sie einmal bei diesem Wert starten und etwas experimentieren.

ASSIGN

Format: ASSIGN [[<name>]<dir>] [LIST] [EXISTS] [REMOVE]

Muster: ASSIGN "NAME, VERZEICHNIS, LIST/S, EXISTS/S, REMOVE/S"

Zweck: Zuordnung eines logischen Geräts an ein Verzeichnis der Dateistruktur.

Pfad: C:ASSIGN

Beschreibung:

NAME ist der Name eines logischen Geräts mit einer Länge von auch mehr als drei Buchstaben, der dem Pfad zu einem VERZEICHNIS zugewiesen wird. Geben Sie nur den Namen ein, wird das entsprechende logische Gerät gelöscht.

Der Befehl ASSIGN alleine oder mit dem Schlüsselwort LIST schreibt die Namen aller aktuellen Geräte auf den Bildschirm. ASSIGN arbeitet immer auf den Namen einer Diskette oder Festplatte bezogen, ohne zu berücksichtigen, in welchem Laufwerk sie eingelegt ist. Alle ASSIGNs werden beim Reboot gelöscht, und müssen neu gesetzt werden.

Wird das Schlüsselwort EXISTS angegeben, so durchsucht ASSIGN die Liste aller Zuweisungen und gibt den Namen des Geräts und das Verzeichnis an, das dem Gerät zugeordnet ist. Wird ein Gerät des geforderten Namens nicht gefunden, so wird der Fehlercode WARN gesetzt.

Mit dem Schlüsselwort REMOVE kann man ein Peripheriegerät von der Liste der Assignments löschen. Es werden dabei weder der Speicher noch die Resourcen, die das Gerät benutzte, freigegeben. *Seien Sie mit dieser Option vorsichtig, wahllose Anwendung der Option kann zu einem Systemabsturz führen!*

Beispiel(e):

```
ASSIGN WORK: DH1:Quelltexte/Superprogramm/Version3_00
```

Der logische Gerätename WORK: wird dem Verzeichnis »DH1:Quelltexte/Superprogramm/Version3_00« zugewiesen. Nun soll die Datei XYZ im Verzeichnis »DH1:-Quelltexte/Superprogramm/Version3_00« auf dem Bildschirm ausgegeben werden. Dazu genügt der folgende Befehl:

```
TYPE WORK: XYZ
ASSIGN LIST
```

Zeigt alle logischen Geräte an. Dabei werden die vollen Verzeichnis-Namen sichtbar.

AVAIL

Format: AVAIL [CHIP|FAST|TOTAL]

Muster: AVAIL "CHIP/S,FAST/S,TOTAL/S"

Zweck: Zeigt die Größe des zur Zeit verfügbaren CHIP- und FAST-Speichers an.

Pfad: C:AVAIL

Beschreibung:

AVAIL liefert einen Überblick über den gesamten verfügbaren RAM-Speicher im System, wobei die Größe des CHIP- wie auch FAST-Speichers angegeben werden. Für jeden Speichertyp gibt AVAIL den Gesamtumfang, den verfügbaren, den davon belegten Bereich und den größten zusammenhängenden Speicherblock an, der noch frei ist.

Bei Verwendung der Schlüsselworte CHIP, FAST und/oder TOTAL zeigt AVAIL jeweils nur die Größe des CHIP-, FAST- oder Gesamt-RAM-Speichers an. Dieser Wert kann für Vergleiche in Befehlsdateien verwendet werden.

Beispiele:

```
AVAIL

Type Available  In-Use  Maximum  Largest
chip    745632  294520  1040152   739888
fast   1406040  691080  2097120  1022640
total  2151672  985600  3137272  1022640

1> AVAIL CHIP 745632
```

Im System war ein FAT-Agnus-Chip mit 1 Mbyte Chip-RAM vorhanden!

Siehe auch: EXECUTE, IF

BINDDRIVERS

Format: BINDDRIVERS

Muster: BINDDRIVERS

Zweck: Einbinden von Gerätetreibern für zusätzliche Hardware.

Pfad: C:BINDDRIVERS

Beschreibung:

Das Kommando BINDDRIVERS gehört in die Startup-Sequenz. Es wird dazu verwendet, Gerätetreiber (z.B. Treiber für eine SCSI-Festplatte), die sich im Verzeichnis mit dem Namen »SYS:Expansion« befinden, in das Betriebssystem einzubinden.

Beispiel(e):

```
BINDDRIVERS
```

BREAK

Format: BREAK <task>[ALL] [C] [D] [E] [F]

Muster: BREAK "PROZESS/A, ALL/S, C/S, D/S, E/S, F/S"

Zweck: Setzen der Attention-Flags.

Pfad: C:BREAK

Beschreibung:

BREAK setzt die einzeln anzugebenden Attention-Flags eines Task. PROZESS ist die Nummer des Prozesses, der unterbrochen werden soll. C setzt das Ctrl+C-Flag, D das Ctrl+D-Flag und so weiter. ALL setzt alle angegebenen Flags. Wird kein Flag angegeben, setzt Amiga-DOS automatisch das Ctrl+C-Flag.

BREAK hat den gleichen Effekt, als würde bei aktiviertem Fenster des Prozeß eine der vier Tastenkombinationen Ctrl+C, D, E oder F eingegeben. Wenn der entsprechende Prozeß eines der vier Flags abfragt, und terminiert, falls eines der Flags gesetzt ist, kann er so unterbrochen werden.

Beispiel(e):

```
BREAK 7
```

setzt das Ctrl+C-Flag in Task 7

```
BREAK 5 D E
```

setzt die Flags Ctrl+D und Ctrl+E in Task 5.

CD

Format: CD[dir]

Muster: CD "DIR"

Zweck: Setzt das aktuelle Verzeichnis.

Pfad: C:CD

Beschreibung:

CD dient dem Festsetzen oder Ändern des aktuellen Verzeichnisses oder Laufwerks. Bei Eingabe von CD ohne Angabe von Parametern wird der Name des aktuellen Verzeichnisses angezeigt. Folgt dem Befehl ein Verzeichnis-Name, bezeichnet er ein neues aktuelles Verzeichnis.

Wird der Name eines physikalischen oder logischen Geräts angegeben, wechselt CD ebenfalls das aktuelle Laufwerk. Um das jeweils übergeordnete Verzeichnis zum aktuellen Verzeichnis zu machen (falls eines existiert), geben Sie nur CD gefolgt von einem einzelnen Schrägstrich (/) ein. Dieser Befehl »CD / benennt das in der Hierarchie nächsthöhere Verzeichnis zum aktuellen, es sei denn, das aktuelle Verzeichnis ist schon das Wurzelverzeichnis und es kann nicht mehr höher gestiegen werden.

Mehrere Schrägstriche hintereinander sind erlaubt. Für jeden Schrägstrich steigt das aktuelle Verzeichnis entsprechend eine Stufe höher.

Beispiel(e):

```
CD DF1:arbeit
```

macht das Verzeichnis »arbeit« auf der Diskette in Laufwerk 1 zum aktuellen Verzeichnis. Das aktuelle Laufwerk wird dadurch ebenfalls DF1: zugewiesen.

```
CD SYS:COM/BASIC
```

und später dann

```
CD /
```

setzt das aktuelle Verzeichnis zu »SYS:COM«.

CHANGETASKPRI

Format: CHANGETASKPRI <priorität> [<Prozeß>]

Muster: CHANGETASKPRI "PRIORITÄT/A,PROZESS/K"

Zweck: Ändern der Priorität von Prozessen, die von der SHELL über den Kommandozeilen-Interpreter gestartet wurden.

Pfad: C:CHANGETASKPRI

Beschreibung:

Der Amiga weist jedem laufenden Prozeß eine Priorität zu, die angibt, welchen Anteil der Systemrechenzeit der Prozeß nutzen darf. Normalerweise laufen die meisten Benutzerprozesse unter der Priorität 0 und teilen die Rechenzeit zu gleichen Teilen. Mit dem vorliegenden Befehl sind Sie nun in der Lage, die Priorität des aktuellen SHELL-Prozesses zu verändern. Alle Prozesse, die von diesem SHELL-Prozeß gestartet werden, behalten diese eingestellte Priorität.

Wird zusätzlich noch die Nummer eines bestimmten Prozesses angegeben, wird dessen Priorität auf den angegebenen Wert geändert. Die Nummern der Prozesse, die von einer SHELL gestartet wurden, können mit dem Befehl STATUS ermittelt werden.

Sie können rein theoretisch Prioritäten-Nummern von –128 bis +127 angeben. Mit zu großen Werten bringen Sie allerdings einige Unordnung in die systeminternen Abläufe, Gerätetreiber wie Trackdisk-Device und der Console-Handler, laufen mit der Priorität +20. Geben Sie einem eigenen Prozeß eine höhere Priorität, so kann es dazu kommen, daß das Betriebssystem nicht mehr rechtzeitig auf Eingaben von einem Peripheriegerät reagieren kann. Begnügen Sie sich deshalb möglichst mit Werten zwischen –5 bis 5.

Eine mögliche Anwendung ist beispielsweise die gleichzeitige Arbeit mit einem Compiler und einem Editor. Hätten beide Prozesse (Compiler und Editor) gleiche Priorität, dann könnten Sie Ihren Text (oder Ihr Programm) nur sehr stockend editieren, da der Editor ständig vom Compiler unterbrochen wird. Geben Sie dem Editor jedoch eine etwas höhere Priorität, dann werden Ihre Tastenbetätigungen bevorzugt bearbeitet. Der Compiler hat dann viel Rechenzeit zwischen Ihren Tastendrücken, die für den Amiga ewig lang auseinanderliegen.

Beispiel(e):

```
CHANGETASKPRI 5
```

Der momentan aktuelle SHELL-Prozeß und alle anderen, die von ihm aus gestartet werden, besitzen Priorität über alle anderen Benutzerprozesse, die mit der voreingestellten Priorität 0 gestartet wurden.

CLOCKPOINTER

Format: CLOCKPOINTER

Muster: CLOCKPOINTER

Zweck: Mauszeiger dient zur Anzeige der Uhrzeit

Pfad: WORKBENCH1.3D:UTILITIES/CLOCKPOINTER

Beschreibung:

CLOCKPOINTER ändert den normalen Mauszeiger (Pfeil) zu einer Digitaluhr, wenn der Arbeitstisch der Workbench aktiv ist. Minuten und Sekunden werden angezeigt, wenn der Mauszeiger zur linken oberen Ecke des Workbenchfensters bewegt wird. Am linken Rand des Workbenchschirms zeigt der Mauszeiger das Datum an.

CMD

Format: CMD <Schnittstelle> <Dateiname> [OPT s|m|n]

Muster: CMD "SCHNITTSTELLE, DATEINAME, OPT/K S | M | N

Zweck: Umleiten der Ausgabe auf eine Schnittstelle in eine Datei.

Pfad: WORKBENCH1.3D:UTILITIES/CMD

Beschreibung:

CMD_Write ist eine Funktion des Betriebssystem, die das Ein-/Ausgabeverhalten des Computers steuert. Durch CMD können Ausgaben auf die serielle oder parallele Schnittstelle, die über CMD_Write erfolgen, in eine Datei umgeleitet werden. Dadurch wird es möglich, zum Beispiel die Druckausgabe in eine Datei aufzunehmen, was das Auffinden von Programmierfehlern oft erleichtert.

»Schnittstelle« kann die Werte »PARALLEL« oder »SERIAL« annehmen, für »Dateiname« darf eine beliebige gültige Dateibezeichnung innerhalb des Dateisystems angegeben werden, außer den Geräten SER: und PAR: (den beiden Schnittstellen, die ja abgefangen werden sollen).

Die Optionen haben folgende Wirkung:

S Unterdrückt alle einleitenden Steuerbefehle (Escape-Sequenzen) des Schreibvorgangs in der Ausgabe. Das ist beim Ausdruck eines Bildschirms sinnvoll, da dabei durch das erste CMD-Write normalerweise ein Reset (Rücksetzbefehl) an den Drucker gesendet wird. Die Ausgabe wird solange vom Druckertreiber zurückgehalten, bis der Reset vollständig ausgeführt ist und der Drucker wieder Daten empfangen kann, damit bei der Übertragung nichts verloren geht.

Wenn CMD verwendet wird, um die Ausgabe in eine Datei umzuleiten, findet diese Verzögerung nicht statt, deshalb muß CMD mit der Option S mitgeteilt werden, diesen Reset-Befehl auszufiltern, falls der Druckertreiber einen Reset sendet.

M Die Einstellung wird ab jetzt für alle weiteren Ausgaben wirksam (MULTIPLE), bis ein BREAK gesendet wurde oder Control-C eingetippt wird.

N Gibt informative Bildschirmmeldungen beim Einsammeln von Ausgaben auf eine Schnittstelle aus.

Wenn CMD von der Workbench aufgerufen wird, so können die obigen Optionen auch über das Feld TOOLTYPES des Icons von CMD an das Werkzeug übergeben werden. An den Inhalt eines Icons kommt man, wenn man das Icon einmal anklickt, und danach den Menüpunkt INFO der Workbench-Menüleiste anwählt (siehe auch ICONX).

Diese Parameter werden von CMD im Feld TOOLTYPES ausgewertet:

DEVICE	=	Entweder parallel oder seriell, Standardwert ist parallel.
FILE	=	Der Name der Zieldatei, Standardwert ist ram:cmd_file.
SKIP	=	TRUE, um Druckerreset auszufiltern. Standardwert ist FALSE, Reset-Befehle werden nicht übersprungen.
MULTIPLE	=	TRUE, um SKIP für die folgenden Ausgaben einzuschalten. Standardwert ist FALSE für nur eine Ausgabe.
NOTIFY	=	TRUE, um Meldungen über die Tätigkeit von CMD zu erhalten. Standardwert ist FALSE, so daß keine Meldungen ausgegeben werden.

COPY

Format: COPY [[FROM] <Name>] [TO] <Name> [ALL] [QUIET] [BUF|BUFFER = <nn>] [CLONE] [DATE] [COM] [NOPRO]

Muster: COPY "FROM, TO/A,ALL/S,QUIET/S,BUF=BUFFER/K, CLONE/S,DATE/S, COM/S,NOPRO/S"

Zweck: Kopieren einer Datei oder eines gesamten Verzeichnis.

Pfad: C:COPY

Beschreibung:

COPY kopiert eine oder mehrere Datein von einem Platz zu einem anderen. COPY fertigt eine Kopie einer Datei oder eines Verzeichnisses an und schreibt diese in eine mit TO genauer bezeichnete Datei oder Verzeichnis. Der alte, vorher bestehende Inhalt der mit TO bezeichneten Datei wird, falls vorhanden, gelöscht. Der Name hinter FROM muß angegeben werden. Wenn ein unter TO spezifiziertes Verzeichnis nicht schon vorhanden ist, legt COPY das benötigte Verzeichnis selbständig neu an.

Soll sich TO auf das aktuelle Verzeichnis beziehen, kann das mit Doppelanführungszeichen »""« angegeben werden.

Hängen Sie den Zusatz ALL an, kopiert COPY alle Dateien aus allen Unterverzeichnissen des FROM-Verzeichnisses. In diesem Fall werden automatisch, soweit nötig, Unterverzeichnisse in dem TO-Verzeichnis geschaffen. Der Name der aktuellen Datei, die

kopiert wird, erscheint während des Vorgangs auf dem Bildschirm. Fügen Sie noch zusätzlich QUIET hinten an, so wird diese Ausgabe der aktuell kopierten Dateien unterdrückt.

Sie können für die Namen der zu kopierenden Dateien auch eine Schablone mit einem oder mehreren Jokern angeben. In diesem Fall kopiert Amiga-DOS jede Datei, die die vorgegebenen Kriterien der Schablone erfüllt. Schlagen Sie bei der Beschreibung des Befehles LIST nach. Dort finden Sie eine vollständige Beschreibung dieser Joker.

Aufgrund der seit DOS 1.2 veränderten Disk-Organisation können Sie die Arbeitsgeschwindigkeit Ihres Rechners verbessern, wenn Sie ab und an Ihre Arbeitsdiskette mit dem Befehl COPY ALL auf eine andere leere Diskette kopieren und dann mit dieser weiterarbeiten.

Das Schlüsselwort CLONE bewirkt, daß neben der Datei das Datum, Kommentare, die mit FILENOTE erstellt wurden, und die Schutzflags der Originaldatei kopiert werden. DATE und COM weisen COPY an, nur das Datum bzw. den Kommentar mitzukopieren. Um zu verhindern, daß die Schutzflags der Originaldatei automatisch mitkopiert werden, kann das Schlüsselwort NOPRO verwendet werden.

Beispiel(e):

```
COPY datei1 TO :arbeit/datei2
```

kopiert die Datei »datei1« aus dem aktuellen Verzeichnis in das Verzeichnis »arbeit« und legt es dort als »datei2« ab.

```
COPY system/#? TO DF1:backup
```

kopiert alle Dateien des Verzeichnisses »:system« in das Verzeichnis »backup« auf der Diskette in Laufwerk DF1:.

```
COPY DF0: TO DF1: ALL CLONE QUIET
```

erstellt eine Kopie der gesamten Diskette inklusive aller Kommentare von Laufwerk 1 auf die Diskette in Laufwerk 2. Dabei werden die Namen der aktuell kopierten Dateien nicht auf dem Bildschirm ausgegeben.

```
COPY test-#? TO DF1:xyz
```

kopiert alle Dateien des aktuellen Verzeichnisses, die mit »test« beginnen, in das bereits auf der Diskette in Laufwerk 1 bestehende Verzeichnis »xyz«.

```
COPY test.datei TO PRT:
```

gibt die Datei »test.datei« auf dem angeschlossenen Drucker aus.

```
COPY * TO NEWCON:10/10/200/100/
```

Nach Eingabe dieses Befehls muß das Eingabefenster angeklickt werden. Dadurch wird es wieder zum Eingabefenster. Jedes Zeichen, das Sie nun eingeben, wird in das neu geöffnete Fenster kopiert. Mit [Ctrl]+[\] wird das zweite Fenster wieder geschlossen.

```
COPY DF0:#?/#? TO DF1: ALL
```

kopiert alle Dateien aus allen Unterverzeichnissen in das Haupt-Verzeichnis auf der Diskette in DF1:.

Siehe auch: JOIN

DATE

Format: DATE [<datum>][<zeit>][TO=VER<name>]

Muster: DATE "DATUM,ZEIT,TO=VER/K"

Zweck: Stellen der Systemuhr oder Anzeigen von Datum/Uhrzeit.

Pfad: C:DATE

Beschreibung:

DATE zeigt das Datum und die Zeit im System an und ermöglicht, sie neu zu stellen. DATE ohne Angabe von Parametern zeigt das aktuelle Datum, den Wochentag und die Zeit des System auf dem Bildschirm. Die Zeit wird im 24-Stunden-Format dargestellt. Bitte verwechseln Sie die Systemuhr nicht mit der eventuell in Ihren Amiga (zum Beispiel Amiga 2000) eingebauten batteriegepufferten Uhr. Zum Übernehmen der mit Date gestellten Uhrzeit in die batteriegepufferte Uhr siehe SETCLOCK.

DATE mit einer Datumsangabe dahinter setzt das Datum. Die Form für das Datum lautet TT-MMM-JJ, für T=Tag, M=Monat (in der englischen Schreibweise) und J=Jahr. Wenn das Datum schon gesetzt ist, kann es noch durch Eingeben eines Wochentagnamens oder durch die Bezeichnungen »Tomorrow« und »Yesterday« (für Morgen und Gestern) verändert werden.

DATE mit einer Zeitangabe ändert die Systemuhrzeit. Die Form der Zeitangabe ist SS:MM, für S=Stunden und M=Minuten. Die Uhrzeit braucht nicht vierstellig angegeben zu werden. Zu beachten ist, daß Amiga-DOS nur dann erkennt, daß die Zeit statt des Datums gemeint ist, wenn ein Doppelpunkt (:) verwendet wird.

Hervorzuheben ist weiterhin, daß Datum und Zeit zusammen oder unabhänig voneinander und in jeder Reihenfolge eingegeben werden können, da DATE die Unterscheidung zwischen Datum und Uhrzeit anhand des Formats der Eingabe trifft.

Haben Sie das Datum nicht gesetzt, dann wird das Datum des Systems automatisch beim Validieren einer neu eingelegten Diskette auf das Datum der zuletzt erzeugten Datei gesetzt, falls diese ein neueres Datum hat wie das aktuelle Systemdatum. Siehe auch Abschnitt 2.5 »Der Aktualisierungsprozeß«.

Mit TO wird die Ausgabe in eine näher spezifizierte Datei geschrieben.

Beispiel(e):

```
DATE
```

zeigt die aktuelle Zeit der Systemuhr an.

```
DATE 6-FEB-89
```

Setzt das aktuelle Datum auf den 6. Februar 1989. Die Zeit wurde nicht gestellt.

```
DATE TOMORROW
```

stellt das Datum einen Tag weiter.

```
DATE TO Zeit
```

schreibt die Systemzeit in die Datei Zeit.

```
DATE 10:50
```

setzt die Systemuhrzeit auf 10.50 Uhr.

```
DATE 01-jan-02
```

setzt das Datum auf den 01. Januar 2002. (Das frühestmögliche Datum ist der 01. Januar 1978).

DELETE

Format: DELETE <name>[<name>*] [ALL] [Q|QUIET]

Muster: DELETE "NAME,,,,,,,,ALL/S,Q=QUIET/S"

Zweck: Löschen einer oder mehrerer Dateien oder Verzeichnisse.

Pfad: C:DELETE

Beschreibung:

DELETE löscht bis zu zehn Dateien oder Verzeichnisse auf einmal. DELETE versucht, jede angegebene Datei zu löschen. Falls eine Datei nicht gelöscht werden kann, wird eine Meldung auf dem Bildschirm ausgegeben, und DELETE versucht, die nächste Datei in der Liste zu löschen.

Verzeichnisse können nur dann gelöscht werden, wenn sich keine Datei mehr darin befindet. Sie können auch einen Joker dazu verwenden, um eine Reihe von Dateien zu bezeichnen. In der Beschreibung des Befehls LIST ist eine vollständige Übersicht der möglichen Joker zu finden. Diese Joker können Gruppen von Dateien oder Verzeichnisse eingrenzen. Alle Dateien, die auf eine Schablone passen, werden gelöscht. Seien Sie also entsprechend vorsichtig damit!

Wird der Zusatz ALL und ein Verzeichnis-Name angehängt, dann löscht DELETE das angegebene Verzeichnis rekursiv mit allen darin enthaltenen Unterverzeichnissen und alle Dateien mit allen darin enthaltenen Verzeichnissen und deren Unterverzeichnissen.

Beispiel(e):

```
DELETE Altes_Bild
```

Löscht die Datei Altes_Bild

```
DELETE arbeit/file1 arbeit/file2 arbeit
```

Löscht die Dateien »file1« und »file2« im Verzeichnis »arbeit«. Anschließend wird das leere Verzeichnis »arbeit« gelöscht.

```
DELETE t#?/#?(1|2)
```

Löscht alle Dateien, deren Namen mit »1« oder »2« enden und die in Verzeichnissen stehen, deren Namen mit »t« beginnen.

```
DELETE DF1:#? ALL
```

Löscht alle Dateien auf der Diskette in Laufwerk DF1:. Siehe auch: DIR (Option DEL).

DIR

Format: DIR[<name>] [OPT A|I|A|D] [ALL] [DIRS] [INTER] [FILES]

Muster: DIR "NAME,OPT/K,ALL/S,DIRS/S,INTER/S,FILES/S"

Zweck: Anzeige eines sortierten Inhaltsverzeichnis eines Geräts oder Verzeichnis der Dateistruktur

Pfad: C:DIR

Beschreibung:

DIR gibt die Dateien eines Verzeichnisses in sortierter Form auf dem Bildschirm aus. DIR kann Dateien aus Unterverzeichnissen beinhalten, und DIR kann auch im interaktiven Dialog benutzt werden.

DIR ohne Parameter zeigt die Dateien im aktuellen Verzeichnis. DIR gefolgt von einem Verzeichnis-Namen zeigt die Dateien des bezeichneten Verzeichnisses. Dabei werden zuerst stets alle Unterverzeichnisse auf dem Bildschirm ausgegeben, gefolgt von einer sortierten, zweispaltigen Dateien-Liste. Diese Ausgabe können Sie mit [Ctrl]+[C] stoppen.

Mit OPT A, oder alternativ dazu dem Schlüsselwort ALL, wird der Inhalt aller Unterverzeichnisse und deren jeweiliger Unterverzeichnisse rekursiv von dem angegebenen Verzeichnis aus ausgegeben. Jede Dateien-Unterliste wird dabei etwas eingerückt.

Sollen nur Verzeichnis-Namen ausgegeben werden, so verwenden Sie die OPT D oder DIRS. OPT I oder INTER legt fest, daß DIR im interaktiven Dialog abläuft. In diesem Fall gibt der Rechner jede Datei und jedes Verzeichnis einzeln aus. Ein Fragezeichen dahinter signalisiert Ihnen, daß Ihr Amiga nun eine Eingabe Ihrerseits verlangt.

Um den nächsten Namen zu erhalten, sollten Sie die Return-Taste betätigen. Möchten Sie den Dialog beenden, so geben Sie einfach »Q« ein. Um zur ursprünglichen Verzeichnis-Stufe zurückzukehren oder um abzubrechen, etwa weil man schon auf der Stufe des ursprünglichen Verzeichnisses ist, gibt man »B« ein.

Ist der angezeigte Name ein Verzeichnis und sollen dessen Dateien und Unterverzeichnisse ausgegeben werden, tippen Sie nur »E«. Sie verwenden »E« und »B«, um die verschiedenen Verzeichnis-Stufen auszuwählen.

Mit dem Befehl »DEL« kann eine Datei oder ein Verzeichnis gelöscht werden. Dies funktioniert aber nur dann, wenn das Verzeichnis keine Einträge mehr enthält. Mit dem Befehl »DEL« ist übrigens die Eingabe der drei Buchstaben D, E und L gemeint, nicht die [DEL]-Taste.

Mit »T« wird die Datei am Bildschirm ausgegeben. [Ctrl]+[C] stoppt die Ausgabe und veranlaßt die Rückkehr in den interaktiven Dialog. Um die mögliche Antwort auf eine interaktive Frage zu finden, geben Sie »?« ein.

Mit »C« oder »COM« können sie während der interaktiven Arbeit mit DIR einen DOS-Befehl abschicken. Nach der Beendigung des Befehls fährt DIR mit der Arbeit an der Stelle fort, die vor der Unterbrechung erreicht wurde. Das Optionflag C und der Befehl lassen sich auch in einer Zeile zusammenfassen, indem der komplette Befehl von Anführungszeichen eingeschlossen wird. Zum Beispiel

```
C "type diskcopy.info HEX"
```

Die Option FILES bewirkt, daß nur Dateien in einem Verzeichnis ausgegeben werden, die Ausgabe von Unterverzeichnisnamen wird unterdrückt.

Beispiel(e):

```
DIR
```

gibt eine Liste aller Dateien des aktuellen Verzeichnisses aus.

```
DIR DF0: ALL
```

listet die gesamte Verzeichnis-Struktur der Diskette im ersten Laufwerk.

DISKCHANGE

Format: DISKCHANGE <Laufwerk>

Muster: DISKCHANGE "Laufwerk/A"

Zweck: Amiga-DOS wird mitgeteilt, daß Sie die Diskette in einem Laufwerk gewechselt haben, das einen Diskettenwechsel nicht automatisch erkennt.

Pfad: C:DISKCHANGE

Beschreibung:

Rufen Sie DISKCHANGE auf, um Amiga-DOS darüber zu informieren, daß Sie eine Diskette im »Laufwerk« gewechselt haben. Dies ist nur bei 5,25-Zoll-Laufwerken notwendig, die keine automatische Rückmeldung vornehmen, sobald eine Diskette gewechselt wurde. Legen Sie also die neue Diskette ein und tippen Sie vor irgendeiner anderen Aktivität:

```
DISKCHANGE df2:
```

Die neue Diskette wird von Amiga-DOS erkannt. Wenn der Name einer Diskette mit RELABEL geändert wurde, kann dies mit DISKCHANGE der Workbench mitgeteilt werden.

DISKCOPY

Format: DISKCOPY [FROM] <disk> TO <disk> [NAME <name>]

Muster: DISKCOPY "FROM/A, TO/A/K, NAME/K"

Zweck: Kopieren einer Amiga-DOS-Diskette.

Pfad: SYSTEM:DISKCOPY

Beschreibung:

DISKCOPY erstellt eine Kopie des gesamten Inhalts der Diskette, die Sie mit FROM bezeichnet haben, und schreibt deren Inhalt auf die mit TO genauer bezeichnete Diskette. DISKCOPY formatiert die neue Diskette gleichzeitig während des Kopiervorgangs.

Normalerweise wird der Befehl zur Erstellung von Sicherheitskopien (Backup) verwendet. DISKCOPY arbeitet mit allen Diskformaten (auch Harddisks oder 5,25-Zoll-Disketten), die vorher mit dem Befehl MOUNT initialisiert wurden. Ziel und Quelle müssen allerdings dasselbe Format haben. Wollen Sie Informationen zwischen unterschiedlich genormten Disketten, wie einer 3,5- und einer 5,25-Zoll-Diskette, kopieren, müssen Sie den COPY-Befehl verwenden. Falls die Recoverable-RAM-Disk RAD: in der »Mountlist« mit 79 Zylindern eingetragen ist, so kann eine 3,5-Zoll-Diskette mit

```
DISKCOPY FROM DF0: TO RAD:
```

in RAD: kopiert werden.

Nachdem Sie den Befehl eingegeben haben, fordert Amiga-DOS Sie auf, die richtigen Disketten einzulegen. Zu diesem Zeitpunkt müssen Sie unbedingt die richtigen Ursprungs- und Zieldisketten einlegen! Der Befehl kann auch zur Erstellung einer Kopie einer Diskette auf einem Einzellaufwerk verwendet werden. Wenn Sie Ursprung und Ziel mit der gleichen Geräte-Nummer belegen, liest das Programm soviel Daten wie möglich von der Ursprungdiskette in den Speicher. Danach wird zum Einlegen der Zieldiskette in das Laufwerk aufgefordert und der Speicherinhalt auf die Zieldiskette kopiert. Dieser Ablauf wird so lange wiederholt wie notwendig.

Wird von Ihnen kein neuer Name für die Zieldiskette angegeben, so benennt DISKCOPY die neue Diskette mit dem Namen der alten. Amiga-DOS kann zwischen zwei Disketten mit gleichlautenden Namen unterscheiden, denn auf jeder Diskette wird das Datum und die Zeit ihrer Erstellung vermerkt. DISKCOPY schreibt der neuen Diskette die aktuelle Datum- und Zeitangabe des Systems als Entstehungszeitpunkt in eine Systemspur.

Seit DOS 1.2 existiert ein neuer Requester (Nachfragemeldung) für DISKCOPY:

```
Diskcopy:      Failed to open icon.library
Retry          Cancel
```

Dieser Requester erscheint, wenn das Kommando DISKCOPY keinen Zugriff auf die »icon.library« auf der Workbench-Disk hat. Das passiert beispielsweise, wenn Sie unter CLI folgendes eingeben:

```
COPY C:DISKCOPY to RAM:
```

(Entfernen Sie die Workbench-Disk)

```
DISKCOPY
```

Jetzt erscheint ein Requester, der Sie auffordert, eine Diskette einzulegen, auf der sich das Verzeichnis LIBS: befindet. Das ist normalerweise die Workbench-Diskette. In diesem Fall legen Sie einfach die Workbench-Diskette ein.

Beispiel(e):

```
DISKCOPY FROM DF0: TO DF1:
```

kopiert die Diskette in Laufwerk 0 komplett auf die Diskette in Laufwerk 1.

```
DISKCOPY FROM DF0: TO DF0:
```

kopiert eine Diskette auf eine andere. Dabei wird nur Laufwerk 0 benutzt.

siehe auch:COPY

DISKDOCTOR

Format: DISCDOCTOR <Laufwerk>:

Muster: DISKDOCTOR "LAUFWERK/A"

Zweck: Versuch der Wiederherstellung zerstörter Disketten.

Pfad: C:DISKDOCTOR

Beschreibung:

Wenn Ihr Amiga eine Diskette mit einem zerstörten Dateisystem in einem Laufwerk vorfindet, so wird das sofort gemeldet (der Inhalt einer Diskette kann zum Beispiel dann zerstört werden, wenn Sie sie aus dem Laufwerk nehmen, obwohl die Laufwerks-LED noch leuchtet). DISKDOCTOR stellt nun so viele Dateien wieder her, wie möglich.

Nachdem eine Diskette mit diesem Befehl bearbeitet wurde, sollten alle darauf befindlichen Dateien schleunigst auf eine Diskette mit unbeschädigtem Dateisystem kopieren werden (nicht mit DISKCOPY, sondern mit COPY!) und die beschädigte Diskette neu formatieren.

DISKDOCTOR arbeitet auch mit dem Fast-Filing-System zusammen. Er kann jedoch nur dann korrekt funktionieren, wenn in der Datei MOUNTLIST im DEVS:-Verzeichnis das Schlüsselwort DOSTYPE 0x444F5301 (Hexadezimal DOS1) für das Gerät, das das Fast-Filing-System benutzt, korrekt eingetragen worden ist. Sonst zerstört DISKDOCTOR den vielleicht noch brauchbaren Inhalt der Festplatte völlig, da sich das alte File-System und Fast-Filing-System im physikalischen Format deutlich unterscheiden.

Beispiel(e):

Amiga-DOS hat eine zerstörte Diskette identifiziert, falls einer der folgenden Requester auf dem Bildschirm erscheint:

```
»Volume Textfiles is not validated«
```

oder:

```
»Error validating disk/Disk is unreadable«
```

Nachdem Sie noch ein paar Leseversuche unternommen haben, geben Sie ein (die Diskette befindet sich in DF0:):

```
DISKDOCTOR DF0:
```

Wenn der Befehl seine Arbeit getan hat, dann erscheint die Meldung:

```
»Now copy files required to a new disk and reformat this disk.«
```

DOWNLOAD

Format: DOWNLOAD [FROM]<name> [TO]<name>

Muster: DOWNLOAD "FROM/A, TO/A"

Zweck: Programme von anderen Rechnern einlesen.

Pfad: Nicht allgemein erhältlich; war Bestandteil des Crossdevelopment-Package auf einer SUN (Workstation von SUN-Microsystems) für den Amiga

Beschreibung:

Der Befehl DOWNLOAD dient dem Empfang von Programmen, die auf anderen Rechnern, wie zum Beispiel einer SUN-Workstation, für den Amiga erstellt wurden, und dann zum Ablaufen auf den Amiga überspielt werden müssen.

DOWNLOAD kann nur in Verbindung mit dem »BillBoard« (Zusatzgerät für die SUN) verwendet werden. Geben Sie zunächst folgenden Befehl auf der SUN ein:

```
binload -p&
```

(Beachten Sie, daß der Befehl binload kein Amiga-DOS-Befehl ist; binload wird nur auf der SUN verwendet.) Geben Sie dann auf dem Amiga den folgenden Befehl ein:

```
DOWNLOAD sun-Dateiname amiga-Dateiname
```

Bevor Sie die SUN starten, sollten das Bill-Board und der Amiga angeschlossen und eingeschaltet sein, denn sie werden sonst von der SUN nicht erkannt. Ferner sollte der SUN-Dateiname mit dem Zusatz ».ld« enden und ein gelinktes Loadfile bezeichnen. Wichtig ist, daß DOWNLOAD-Dateien immer relativ zu dem SUN-Verzeichnis verwaltet werden, in dem BINLOAD gestartet wurde. Wenn man sich nicht mehr an das Verzeichnis erinnern kann, in dem BINLOAD gestartet wurde, dann muß der volle Zugriffspfad angegeben werden.

BINLOAD verhält sich auf der SUN wie jedes übliche Werkzeug des Betriebssystems UNIX.

Die voreingestellte Ausgaberichtung ist das Fenster, in dem BINLOAD gestartet wurde. DOWNLOAD kann mit [CTRL]+[C] abgebrochen werden.

Nahezu kein normaler Anwender wird Amiga-Programme auf einer SUN-Workstation entwickeln (wer hat schon eine SUN), so daß DOWNLOAD nicht auf der Workbench mitgeliefert wird.

Beispiel(e):

```
binload -p&   (Auf der SUN)
DOWNLOAD test.ld test (auf dem Amiga)
```

oder

```
DOWNLOAD /usr/fred/DOS/test.ld test
```

Diese Befehle laden die angegebenen Dateien von der SUN zum Amiga.

FIXFONTS

Format: FIXFONTS

Muster: FIXFONTS

Zweck: Bringt die ».font« Dateien im FONTS:-Verzeichnis auf den neuesten Stand.

Pfad: SYSTEM:FIXFONTS

Beschreibung:

Amiga-DOS ist eine Portierung von dem Betriebssystem TRIPOS auf den Amiga, das für das Diskettenhandling auf den Gerätetreiber Trackdisk-Device aufsetzt. Durch das nicht optimal aufeinander abgestimmte Design dieser zwei Komponenten des Betriebssystems kommt es zu dem etwas ineffektiven Verhalten des Amigas beim Laden von Daten von einer Diskette.

Ursprünglich war Amiga-DOS als reines Diskettenbetriebssystem gedacht, und wurde deshalb für optimale Arbeitsgeschwindigkeit mit den recht langsamen Laufwerken (dank TRIPOS) hingefriemelt. Um die vorhandenen Zeichensätze auf einer Diskette schnell ausfindig zu machen, ohne das gesamte Inhaltsverzeichnis von FONTS: einlesen zu müssen, wurden ».font«-Dateien in das Verzeichnis FONTS: geschrieben, in denen eingetragen ist, welche Größen eines Zeichensatzes vorhanden sind.

Sollte nun ein Zeichensatz bestimmter Größe neu hinzukopiert oder ein anderer gelöscht werden, werden die Zeichensätze nicht mehr korrekt gefunden, da vorher noch der entsprechende .font-Eintrag in Ordnung gebracht werden muß.

FIXFONTS durchsucht alle Unterverzeichnisse im FONTS:-Verzeichnis und erstellt, falls notwendig, neue korrekte .fonts-Einträge, in denen die eingetretenen Änderungen im FONTS:-Verzeichnis festgehalten werden.

ECHO

Format: ECHO <Zeichenfolge> [NOLINE] [FIRST <nn>] [LEN <nn>]

Muster: ECHO "ZEICHENFOLGE, NOLINE/S,FIRST/S,LEN/S"

Zweck: Eine Zeichenfolge ausgeben.

Pfad: C:ECHO

Beschreibung:

ECHO gibt den Text aus, den Sie als Argument übergeben. Es schreibt diesen Text auf das aktuelle Ausgabegerät, also eine Datei oder ein Gerät; normalerweise auf den Bildschirm. ECHO ist meist nur innerhalb einer Scriptdatei oder als Teil eines RUN-Befehls sinnvoll. Wird das Argument falsch eingegeben, wird eine Fehlermeldung ausgegeben.

NOLINE verhindert die Ausgabe eines Zeilenvorschub/Wagenrücklauf am Ende der Zeichenfolge. Die Optionen FIRST und LEN dienen zur Ausgabe eines Teils der Zeichenfolge. Durch FIRST <nn> wird die Position des ersten Zeichens in der Zeichenfolge angegeben, bei der die Ausgabe beginnen soll. Mit LEN <nn> wird die Anzahl der Zeichen festgelegt, die mit ECHO ausgegeben werden sollen, gerechnet ab dem ersten Zeichen der Zeichenkette.

Fehlt der Parameter FIRST, so werden mit der Option LEN die nn am weitesten rechts stehenden Zeichen der gesamten Zeichenkette ausgegeben.

Die Ausgabe der Zeichenfolge kann mit folgenden Steuerzeichen beeinflußt werden:

*N Zeilenvorschub, der Rest des Textes wird in die nächste Zeile gedruckt.
*E Escape-Zeichen (Dezimalcode 27) um ANSI-Steuersequenzen einzuleiten.
** Druckt einen Stern, der sonst als Sonderzeichen interpretiert würde.

Beispiel(e):

```
ECHO *E[1m   ;Fettschrift ein
ECHO *E[0m   ;Fettschrift aus
```

schaltet Fettschrift im SHELL-Fenster ein und aus.

```
RUN COPY :arbeit/program TO DF1:arbeit ALL +
ECHO "Kopie fertig"
```

erzeugt einen neuen SHELL-Prozeß, kopiert in diesem Prozeß das angegebene Verzeichnis und gibt anschließend die Meldung »Kopie fertig« aus. Dieser Prozeß läuft als Hintergrund-Prozeß ab. Erstellen Sie nun folgende Script-Datei mit dem Namen filekopie (die Dateien ABC und XYZ werden auch benötigt!):

```
ECHO "Start der Script-Datei"
COPY DF0:ABC TO RAM: ABC
COPY DF0:XYZ TO RAM:XYZ
ECHO "Entnimm die Diskette in DF0:"
ECHO "Lege eine neue Diskette in DF0:"
```

```
WAIT 10 SECS
COPY RAM:ABC TO DF0:ABC
COPY RAM:XYZ TO DF0:XYZ
ECHO "FERTIG!"
```

dann führt der Befehl

```
EXECUTE filekopie
```

zur Kopie der angegebenen Dateien auf die RAM-Disk und von dort auf eine neue Diskette.

ENDCLI

Format: ENDCLI

Muster: ENDCLI

Zweck: Ein SHELL-Fenster schließen.

Pfad: C:ENDCLI

Beschreibung:

ENDCLI schließt eine SHELL oder CLI. Amiga-DOS erlaubt ENDCLI nur als interaktiven Befehl (also nicht in Script-Dateien). ENDCLI sollten Sie unter keinen Umständen außerhalb einer SHELL, die mit dem Befehl NEWSHELL oder NEWCLI gestartet wurde, verwenden.

Wenn die einzige SHELL beendet wurde, ohne das die Workbench gestartet ist, oder vorher keine weitere SHELL durch den Befehl NEWSHELL oder NEWCLI geöffnet wurde, beendet ENDCLI die Arbeit mit dem Rechner, nur noch ein Reboot hilft. Der Befehl ENDCLI hat keine Argumente.

Beispiel(e):

```
NEWSHELL
DIR
ENDCLI
```

eröffnet eine neue SHELL, listet das aktuelle Verzeichnis auf und schließt die SHELL wieder.

FAULT

Format: FAULT [<n>*]

Muster: FAULT ",,,,,,,,"

Zweck: Ausgabe von Fehlermeldungen im Klartext.

Pfad: C:FAULT

Beschreibung:

Amiga-DOS liefert die Fehlermeldungen, deren Codes eingegeben wurden, im Klartext. Bis zu zehn Fehlercodes können nacheinander eingegeben werden. Der Befehl erspart Ihnen also lästiges Nachschlagen.

Beispiel(e):

```
FAULT 222
```

Schreibt auf den Bildschirm:

```
FAULT 222: file is protected from deletion
FAULT 221 103 121 218
```

ergibt folgende Meldungen:

```
Fault 221: disk full
Fault 103: insufficient free store
Fault 121: file is not an object module
Fault 218: device (or volume) not mounted
```

FASTMEMFIRST

Format: FASTMEMFIRST

Muster: FASTMEMFIRST

Zweck: Neuordnung der systeminternen Speicherliste.

Pfad: SYSTEM:FASTMEMFIRST

Beschreibung:

Grundsätzlich gibt es in Amiga-Computern zwei verschiedene Sorten Speicher, CHIP-RAM und FAST-RAM. CHIP-RAM wird der Teil des Speichers genannt, auf den die Customchips für Sound und Grafik per Direct Memory Access (DMA) zugreifen können. FAST-RAM dagegen ist Speicher, der außerhalb des Adreßbereichs der Customchips liegt und auf den der Prozessor alleinigen Zugriff hat.

Beim Zugriff auf das CHIP-Memory müssen sich der MC68000-Prozessor und die Customchips die Zeit, in der sie nacheinander auf diesen RAM-Bereich zugreifen können, teilen, wobei aber die Custom-Chips Vorrang haben, wenn sie zum Beispiel komplizierte Grafiken auf dem Bildschirm darstellen müssen. Das bedeutet, daß der Prozessor möglicherweise für einige Taktzyklen den Zugriff auf das CHIP-RAM verwehrt bekommt. Das verlangsamt die Ausführung von Programmen ein wenig.

Steht der Programmcode dagegen im FAST-RAM (so Speichererweiterung vorhanden), kann der Prozessor von den Customchips ungebremst auf das RAM zugreifen.

Es gibt bei zwei Modellen der Amiga-Familie eine Besonderheit in Bezug auf die Verwaltung des internen RAM-Speichers. Der ursprüngliche, heute nicht mehr produzierte Amiga 1000 und der Amiga 2000 A, der im Prinzip ein um Steckplätze erweitertes Motherboard des Amiga 1000 enthält, basieren noch auf dem alten Design der Customchips, das eine

klare Trennung zwischen FAST-RAM und CHIP-RAM macht. Beim neu entwickelten Amiga 500 und dem Amiga 2000 B, deren beider Architektur eine neuere Version der Customchips zu Grunde liegt, gibt es drei Sorten RAM.

Das kommt so: In Hinblick darauf, daß es eigentlich ganz schön wäre, mehr als 512 Kbyte CHIP-RAM zur Verfügung zu haben, vielleicht ein Mbyte, wurde das Design der Chips und des Motherboards darauf vorbereitet, das entsprechende Chips einmal verfügbar sein werden. Im Amiga 500 als interne Erweiterungsplatine vorgesehen, und Amiga 2000 B auf dem Motherboard bestückt, können die Customchips im Moment ein Mbyte CHIP-RAM verwalten, aber noch nicht benutzen. In beiden genannten Rechnern gibt es also 512 Kbyte CHIP-RAM und 512 Kbyte RAM, das kein echtes FAST-RAM ist, sondern dem Bustiming des DMA der Custom-Chips unterliegt.

Beim Zugriff auf diese 512 Kbyte »SLOW«-RAM, die nur der MC68000-Prozessor benutzen kann, solange das sogenannte Enhanced Chip Set (die neuen Custom-Chips mit unter anderem ein Mbyte CHIP-RAM) noch nicht verfügbar ist, wird der Prozessor aber ähnlich wie beim CHIP-RAM gebremst, wenn die Customchips intensiv über DMA auf das eigentliche CHIP-RAM zugreifen, da der SLOW-RAM-Bereich quasi »im voraus« als CHIP-RAM gewertet wird.

Dieses »SLOW«-RAM liegt im Rechnerspeicher ab der physikalischen Adresse $C00000.

Die Autokonfigurationsroutine des Betriebssystems sucht beim Reboot des Rechners unter anderem nach Speichererweiterungen, und ordnet sie nach dem CHIP-RAM in die Systemspeicherliste ein. Dabei wird das »SLOW«-RAM direkt nach dem CHIP-RAM eingeordnet, noch vor echtem FAST-RAM am Expansion-Slot des Amiga 500 und 1000 oder in einem der Steckplätze eines Amiga 2000. Das heißt, Programme werden in das gebremste RAM geladen, anstatt das ungebremste FAST-RAM zu benutzen.

FASTMEMFIRST ändert die Reihenfolge des Speichers in der Systemliste, damit das FAST-RAM vor dem SLOW-RAM benutzt wird, und so die Arbeitsgeschwindigkeit des Rechners leicht erhöht wird. Grafik- und Sounddaten werden natürlich weiterhin in das CHIP-RAM geladen, wenn das Programm diesen Speichertyp vom Betriebssystem korrekt verlangt.

Falls Ihr Amiga ein sehr neues Modell ist (Revision 6.x der Hauptplatine), ist es möglich, daß er schon eine neue Version des CHIP Fat-Agnus enthält, die das eine Megabyte RAM auf der Hauptplatine komplett als CHIP-RAM verwaltet und nutzt. Alle Funktionen dieses Chips sind mit der alten Agnus kompatibel, man hat einfach nur ein Megabyte CHIP-RAM, das alle Programme nutzen können.

Wenn das eine Megabyte RAM auf der Hauptplatine des Amiga als voll nutzbares CHIP-RAM verwaltet wird, wird FASTMEMFIRST natürlich überflüssig.

FILENOTE

Format: FILENOTE [FILE] <file> COMMENT <kommentar>

Muster: FILENOTE "FILE/A,COMMENT/K"

Zweck: Anhängen einer Dateinotiz an eine Datei.

Pfad: C:FAULT

Beschreibung:

FILENOTE ordnet einer gekennzeichneten Datei einen Kommentar zu. Das Schlüsselwort COMMENT leitet einen bis zu 79 Zeichen langen Kommentar ein. Ein Kommentar kann mehr als ein Wort umfassen, das heißt, er darf Leerzeichen zwischen den Worten oder Zeichen enthalten. In diesem Fall muß der Kommentar zwischen Anführungsstrichen (") stehen. Ein Kommentar ist mit einer speziellen Datei verknüpft. Untersuchen Sie diese Datei mit dem Befehl LIST, erscheint eine Zeile tiefer der Kommentar:

```
LIST

prog  30 rwed--- Today 11:07:33 : version 3.3 - 23. März 85
```

Eröffnen Sie eine neue Datei, so ist ihr kein Kommentar angefügt. Wenn Sie eine bestehende Datei, der ein Kommentar zugeordnet ist, überschreiben, so wird der Kommentar beibehalten, selbst wenn der Inhalt der Datei geändert wurde. Der Befehl COPY ... CLONE kopiert die Datei zusammen mit dem Kommentar.

Beispiel(e):

```
FILENOTE programm2 COMMENT "Version 3.3 vom 30. März 85"
```

hängt der Datei Programm2 den Kommentar »Version 3.3 vom 30. März 85« an.

Siehe auch: LIST, COPY

FF

Format: FF [-0] [-N] [FONT]

Muster: FF "-0/S,-N/S,FONT"

Zweck: Erhöhung der Geschwindigkeit des Amiga bei Textausgabe.

Pfad: C:FF

Beschreibung:

FF (Fast-Fonts) ist ein von Charlie Heath von der Firma Microsmiths, Inc. geschriebenes Programm. FF beschleunigt die Ausgabe von Texten in Fenstern, zum Beispiel dem SHELL-Fenster. Da in der Version 1.3 des Betriebssystems noch keine gründliche und tiefgreifende Veränderung am Kickstart-ROM vorgenommen wurde, müssen Patches (Korrekturen) für Betriebssystemroutinen von Diskette nachgeladen werden, und möglichst schon während des Reboot in das System eingebunden werden (siehe: Startup-Sequence Kapitel 4).

Um Fast-Fonts zu aktivieren, ist die Option –0 zu verwenden. Soll Fast-Fonts abgeschaltet werden, ist die Option –N zu verwenden. Ferner kann der Benutzer auch einen Zeichensatz angeben, der statt des Standardzeichensatzes Topaz verwendet werden soll:

```
FF »name«.font
```

Allerdings unterstützt Fast-Fonts nicht die Verwendung von proportionalen Zeichensätzen, bei denen die Breite der einzelnen Buchstaben unterschiedlich ist.

Beispiel(e):

```
FF pearl.font
```

aktiviert Fast-Fonts und lädt statt des Standardzeichensatzes Topaz 80 den Zeichensatz »Pearl«, der auf der Fish-Disk Nr. 61 (verbreitete Diskettenserie mit Public Domain Software) zu finden ist. (Der Zeichensatz muß auf die Workbench kopiert werden, danach muß FIXFONTS den Inhalt des Font:-Verzeichnisses korrigieren. Wegen eines Fehlers im Zeichensatz muß er vor dem Benutzen in den Font-Editor FED aus der SYSTEM:-Schublade geladen werden, und dann in dem Format dieses Programmes mit Hilfe des Menüpunktes SAVE auf die Workbench zurückgeschrieben werden).

FORMAT

Format: FORMAT DRIVE <drivename> NAME <diskettenname> [NOICONS] [QUICK] [FFS] [NOFFS] [INHIBIT]

Muster: FORMAT "DRIVE/A/K, NAME/A/K, NOICONS, QUICK, FFS, NOFFS, INHIBIT"

Zweck: Erstellen einer für den Amiga verwendbaren Diskette.

Pfad: SYSTEM:FORMAT

Beschreibung:

FORMAT formatiert eine neue Diskette oder eine Festplatte durch Anlegen von Spuren und Sektoren, die von Amiga-DOS benötigt werden. Zugleich werden Verzeichnis und Kennzeichnung für belegte und unbelegte Blöcke angelegt. Nur eine formatierte Diskette erhält den gewünschten Namen und kann nun beschrieben und wieder gelesen werden. Beachten Sie bitte, daß die Schlüsselwörter DRIVE und NAME immer angegeben werden müssen. Zulässige Laufwerkkennungen sind: DF0:, DF1:, DF2:, DF3:. Als Diskettenname ist jede beliebige Zeichenkette (String) mit bis zu dreißig Zeichen, aus einem oder mehreren Worten bestehend, erlaubt. Bei mehreren Worten muß der gesamte Name zwischen Anführungsstriche gesetzt werden.

Achtung! FORMAT löscht den Inhalt einer bereits beschriebenen Diskette ohne weitere Warnung. Vergewissern Sie sich, daß keine wichtige Datei auf der zu formatierenden Diskette ist. FORMAT funktioniert bei allen Disketten und Festplatten-Partitionen (eingeschlossen 5,25-Zoll-Disketten), sofern sie, falls notwendig mit dem MOUNT-Kommando initialisiert wurden. FORMAT richtet einen Trashcan (Mülleimer) auf der formatierten Disk ein, falls Sie nicht den Zusatz NOICONS angegeben haben. Mittels [Ctrl]+[C] können

Sie den Formatierungsprozeß auch vorzeitig abbrechen, die Diskette ist dann allerdings unbrauchbar.

Erstellen Sie eine Kopie einer ganzen Diskette mit dem Befehl DISKCOPY, braucht die Zieldiskette übrigens vorher nicht formatiert zu werden.

Um die Zeit für das Neu-Formatieren einer gesamten Diskette abzukürzen, kann eine bereits formatierte Diskette mit der Option QUICK formatiert werden. QUICK überschreibt nur den Boot-Block, das Wurzelverzeichnis der Diskette und die Sektorbelegungstabelle. Das bedeutet, das im Gegensatz zum Formatieren ohne QUICK nicht der Inhalt der gesamten Diskette gelöscht wird, sondern komplett erhalten bleibt. Der Inhalt ist jedoch endgültig verloren, da ohne das Inhaltsverzeichnis und die alte Sektorbelegungstabelle die Diskette wie eine völlig leere Diskette aussieht (mit einem Diskettenmonitor wie Discovery von Ralph Babel sind allerdings die alten Daten noch lesbar).

FFS und NOFFS beziehen sich auf das Fast-Filing-System, das unter Workbench1.3 nur für Festplatten nutzbar ist. Um das Fast-Filing-System zu nutzen, sollte in dem Eintrag in der Datei DEVS:Mountlist für die Partition der Festplatte das Schlüsselwort

```
DosType = 0x444F5301 /* notwendig für FFS */
```

eingetragen sein. Format prüft den Eintrag der Mountlist und verwendet das FastFile-System, falls der Eintrag »DosType = 0x444F5301« vorhanden ist, wenn nicht, wird das alte File-System benutzt. Mit FFS und NOFFS läßt sich dieser Eintrag gezielt überschreiben. FFS veranlaßt die Festplatte mit dem Fast-Filing-System zu formatieren, NOFFS formatiert mit dem alten File-System.

INHIBIT verhindert, daß vor dem Formatieren der Diskvalidator aufgerufen wird. Das ist bei Disketten sinnvoll, die den Amiga bei Validier-Versuchen abstürzen lassen (das gibt's leider auch).

Beispiel(e):

```
FORMAT DRIVE DF0: NAME Arbeitsdiskette
```

formatiert die Diskette in Laufwerk DF0: und gibt ihr den Namen »Arbeitsdiskette«. Das Kommando INITIALIZE ist der FORMAT entsprechende Workbench-Menüpunkt. Genauso wie FORMAT arbeitet er mit allen Disktypen, die mit MOUNT initialisiert wurden. INITIALIZE kopiert einen Mülleimer auf die Diskette. Sollten Sie gerade Ihre SYS:-Diskette in einem Laufwerk haben, dann wird das dafür zuständige Icon von dieser Diskette genommen. Anderenfalls muß die Workbench-Diskette dafür herhalten. Auch INITIALIZE können Sie mit [Ctrl]+[C] jederzeit abbrechen.

Siehe auch: DISKCOPY, INSTALL, RELABEL, Kapitel 5 »Mountlist«

GETENV

Format: GETENV <Name>

Muster: GETENV "NAME/A"

Zweck: Anzeige des aktuellen Werts einer Umgebungsvariablen.

Pfad: C:GETENV

Beschreibung:

GETENV dient zur Anzeige des Inhalts einer Umgebungsvariablen. Umgebungsvariablen werden im ENV:-Verzeichnis gespeichert, das in der Datei S:Startup-Sequence mit

```
MAKEDIR irgend:ein/sinnvoller/Pfad/ENV ;falls nötig
ASSIGN ENV: irgend:ein/sinnvoller/Pfad/ENV ;z.B. RAM:ENV
```

initialisiert werden sollte.

Sie können viele Amiga-DOS-Befehle mit Umgebungsvariablen verwenden, wie z.B. DIR, LIST, ASSIGN, MORE etc. Dies verschafft dem Benutzer große Flexibilität beim Arbeiten mit Umgebungsvariablen. Es gibt wenige professionelle Programme, die Umgebungsvariablen unterstützen, zum Beispiel die WSHELL von William S. Hawes, ein sehr gutes Konkurrenzprodukt für die SHELL des Amiga-DOS, das eine stark erweiterte Funktionalität gegenüber der DOS-SHELL besitzt.

Siehe auch:SETENV

GRAPHICDUMP

Format: GRAPHICDUMP [TINY | SMALL | MEDIUM | LARGE | x-Punkte:y-Punkte]

Muster: GRAPHICDUMP "TINY | SMALL | MEDIUM | LARGE | x-Punkte:y-Punkte"

Zweck: Ausdruck des vordersten SCREEN (virtuellen Bildschirm)

Pfad: WORKBENCH1.3D:UTILITIES/GRAPHICDUMP

Beschreibung:

Der Bildschirm im Vordergrund wird als Grafik auf den Drucker ausgegeben. Dies erfolgt zirka zehn Sekunden nach dem Doppelklicken auf das Icon oder das Eintippen des Befehls auf der SHELL, die Verzögerung erlaubt den für den Ausdruck gewünschten Bildschirm in den Vordergrund zu holen.

Die Größe des Ausdrucks kann durch eine der fünf Optionen festgelegt werden:

TINY — Die Breite des Ausdrucks ist etwa ein 1/4 der maximalen Druckbreite des Druckers, die Höhe des Ausdrucks wird im korrekten Verhältnis zur Breite je nach Auflösung des Screens festgelegt.

SMALL — Der Ausdruck ist etwa halb so breit wie die maximale Druckbreite des Druckers, die Höhe wird dazu im korrekten Verhältnis bestimmt.

MEDIUM — Der Ausdruck nutzt 3/4 der Druckbreite.

LARGE	Maximale Größe.
x-Punkte:y-Punkte	Angabe der absoluten Maße. Durch x-Punkte wird die Breite in Punkten und durch y-Punkte die Höhe in Punkten angegeben. Dies erlaubt jede Größe und jedes Längenverhältnis des Bildes innerhalb der Möglichkeiten des Druckers frei zu wählen.

Damit GRAPHICDUMP die Optionen TINY, SMALL, MEDIUM und LARGE ausführen kann, muß LIMITS im zweiten Menü für Grafikdruck in PREFERENCES auf IGNORE gesetzt sein, ansonsten wird der dort angegebene Wert benutzt.

INFO

Format: INFO [<GERÄT>]

Muster: INFO "GERÄT"

Zweck: INFO erteilt Informationen über die Dateiverwaltung.

Pfad: C:INFO

Beschreibung:

Dieser Befehl zeigt in einer Bildschirmzeile Informationen über jedes Fesplatten- und Diskettenlaufwerk an. Dazu gehören die maximale Speicherkapazität der Diskette in Kilobyte nnnK oder Megabyte nnnM, der momentan belegte und der freie Speicherplatz in Kilobyte (Megabyte) und die freie und belegte Kapazität in Prozent, die Anzahl der Fehler, die aufgetreten sind, der Disketten-Status und der Name der Diskette.

Gibt man optional den Namen eines Geräts an, so werden nur Informationen über dieses Gerät ausgegeben.

Beispiel(e):

```
INFO
```

ergibt folgende Bildschirmausgabe:

```
Mounted disks:
Unit   Size  Used  Free Full Errs  Status  Name
DF1:   No disk present
DF0:   No disk present
RAM:   173K    347      0 100%  0 Read/Write RAM DISK
DH0:    20M  21392  19566  52%  0 Read Only  FAST
DH1:    20M  21263  19695  51%  0 Read/Write USR
Volumes available:
RAM DISK [Mounted]
USR [Mounted]
FAST [Mounted]
```

INITPRINTER

Format: INITPRINTER

Muster: INITPRINTER

Zweck: Laden des Druckertreibers ins RAM.

Pfad: SYSTEM:INITPRINTER

Beschreibung:

INITPRINTER lädt vom Verzeichnis DEVS: die Gerätetreiber parallel- oder serial-device als Schnittstellentreiber, und das darauf aufsetzende Printer-Device.

Durch das Senden von Escape-Codes werden am Drucker die mit dem Programm PREFERENCES festgelegten Voreinstellungen vorgenommen.

INSTALL

Format: INSTALL [DRIVE] DF<n>: [NOBOOT] [CHECK]

Muster: INSTALL"DRIVE/A, NOBOOT/S, CHECK/S"

Zweck: Installieren oder Verändern der Boot-Sektoren auf einer Amiga-DOS-Diskette.

Pfad: C:INSTALL

Beschreibung:

INSTALL macht eine formatierte Diskette zu einer bootfähigen Diskette. Nach INSTALL geben Sie einfach die Bezeichnung des Laufwerks an, in dem sich die Diskette befindet, auf die die Boot-Sektoren geschrieben werden sollen. Die vier möglichen Laufwerksbezeichnungen sind: DF0:, DF1:, DF2: und DF3:.

Mit der Option NOBOOT werden die Boot-Sektoren gelöscht, so daß kein Reboot mit der Diskette mehr erfolgen kann. NOBOOT bewirkt, daß INSTALL die Boot-Sektoren auch auf Disketten schreibt, die nicht als Amiga-DOS-Disketten anerkannt werden.

CHECK überprüft, ob auf der Diskette die original Commodore Boot-Sektoren stehen, oder ob irgendein anderer Code vorhanden ist, und meldet dies, außerdem wird der Rückgabewert in diesem Fall auf WARN gesetzt. Das ist nützlich, wenn man den Verdacht hat, daß auf einer Diskette ein Virus ist, der sich in den Boot-Sektoren verbirgt.

Beispiel(e):

```
INSTALL DF0:
```

macht die Diskette in Laufwerk DF0: zu einer Startdiskette.

```
INSTALL DF0: CHECK
```

hat einen fehlerhaften Boot-Sektor gefunden:

```
May not be standard V1.2/1.3 bootblock
```

JOIN

Format: JOIN <name> <name> [<name>*] AS|TO <name>

Muster: JOIN ",,,,,,,,,,,,,,TO=AS/A/K"

Zweck: JOIN verkettet bis zu 15 Dateien zu einer neuen Datei.

Pfad: C:JOIN

Beschreibung:

Amiga-DOS kopiert die bezeichneten Dateien in der angegebenen Reihenfolge hintereinander in die als AS definierte neue Datei. Der Name der neuen Datei darf mit keinem der eingelesenen Dateinamen übereinstimmen. Die eingelesenen Originaldateien bleiben unverändert, während die neuentstandene Datei eine Kopie aller eingelesenen Dateien beinhaltet. Der Bezeichner TO kann gleichbedeutend zu AS verwendet werden.

Beispiel(e):

```
JOIN teil1 teil2 AS textfile
```

kopiert die angegebenen Dateien »teil1« und »teil2« in die neue Datei »textfile«. Die Originaldateien bleiben unverändert bestehen.

LIST

Format: LIST [<Verzeichnis|Muster>] [P|PAT <Muster>] [KEYS] [DATES] [NODATES] [TO <Dateiname>] [SUB <Subname>] [SINCE <Datum>] [UPTO <Datum>] [QUICK] [BLOCK] [NOHEAD] [FILES] [DIRS] [LFORMAT <Zeichenfolge>]

Muster: LIST "DIR,P=PAT/K, KEYS/S, DATES/S, NODATES/S, TO/K, SUB/K, UPTO/K, QUICK/S, BLOCKS/S, NOHEAD/S, FILES/S, DIRS/S, LFORMAT/K"

Zweck: LIST untersucht und listet bestimmte Informationen über ein Verzeichnis oder eine Datei auf.

Pfad: C:LIST

Beschreibung:

Der erste Parameter, den LIST akzeptiert, ist DIR. Er bietet drei Möglichkeiten. Erstens: DIR kann ein Dateiname sein, dann zeigt LIST die Datei-Information für diese eine Datei an. Zweitens kann DIR ein Verzeichnis-Name sein. Hier zeigt LIST die Datei-Informationen der Dateien und Unterverzeichnisse innerhalb des bezeichneten Verzeichnisses an. Geben Sie gar keinen Namen für DIR an, zeigt LIST den Inhalt des aktuellen Verzeichnisses. Weitere Einzelheiten über das aktuelle Verzeichnis siehe auch unter dem Befehl CD. LIST sortiert übrigens nicht wie der Befehl DIR das Verzeichnis, bevor es auf dem Bildschirm aufgelistet wird. Der Befehl LIST, ohne weitere Angaben, zeigt folgendes an:

```
Name Größe Status Datum Zeit :Kommentar
```

Diese Angaben bedeuten:

Name:	Name der Datei oder des Verzeichnisses.
Größe:	Die Länge der Datei in Bytes. Wenn sich nichts in der Datei befindet, wird »empty« für leer gemeldet. Verzeichnisse werden mit »dir« gekennzeichnet.
Status:	Bezeichnet den verfügbaren Zugriff auf diese Datei in der Form »rwedspa«. Siehe PROTECT zur Erklärung der Funktion der einzelnen Schutzflags.
Datum/Zeit:	Datum und Uhrzeit, zu der diese Datei angelegt wurde.
Kommentar:	Der mit FILENOTE angehängte Kommentar. Er beginnt immer mit einem Doppelpunkt (:).

Soll eine Untermenge der Dateien in einem Verzeichnis ausgewählt werden, kann dies durch Angabe eines Musters anstatt eines eindeutigen Dateinamens geschehen. Eine Suchschablone enthält eine Anzahl Steuerzeichen mit spezieller Bedeutung. Die speziellen Zeichen sind: « ( ) ? % # und |.

Um die spezielle Wirkung dieser Zeichen aufzuheben, muß ihnen ein Apostroph (') vorangestellt werden. So hebt '? die spezielle Wirkung des Zeichens ? und '« des Zeichens« auf.

?	Platzhalter für ein beliebiges einzelnes Zeichen.
%	entspricht einem Nullstring.
#<p>	entspricht null oder mehreren Wiederholungen des Suchmusters <p>.
<p1> <p2>	Aneinanderreihung zweier Suchmuster.
<p1>|<p2>	Entweder paßt Suchmuster <p1> oder <p2>, aber nicht beide.
(und)	Umklammerung mehrerer alternativer Suchmuster, die von einem | (oder) getrennt werden.

Also:

```
LIST PAT A#BC
```

listet AC, ABC, ABBC und so weiter.

```
LIST PAT A#(B|C)D
```

listet AD, ABD, ABCD und so weiter.

```
LIST PAT A?B
```

listet AAB, ABB, ACB und so weiter.

```
LIST PAT A#?B
```

listet AB, AXXB, AZXQB und so weiter.

```
LIST PAT '?#?'#
```

listet ?#, ?AB#, ??## und so weiter.

```
LIST PAT A(B|%)#C
```

listet A, ABC, ACCC und so weiter.

```
LIST PAT #(AB)
```

listet AB, ABAB, ABABAB und so weiter.

KEYS	gibt die Blocknummer jedes Dateikopfes oder Verzeichnisses an.
DATES	gibt das Datum in der Form TT-MMM-JJ mit an. Diese Option gehört zur Grundeinstellung, außer es wird zusätzlich QUICK verwendet.
NODATES	unterdrückt die Ausgabe von Datum und Zeit.
TO	<Dateiname> bezeichnet eine Datei, in die das Ergebnis von LIST geschrieben werden soll. Falls weggelassen, geht die Ausgabe an das aktuelle SHELL-Fenster.
SUB	<Subname> bezeichnet eine Teilzeichenkette (Substring), die in dem Datei- oder Verzeichnisnamen vorkommen muß.
SINCE	<Datum> listet nur Dateien auf, die am bezeichneten Datum oder später bearbeitet wurden. Das Datum kann in der Form TT-MMM-JJ oder als Wochentag der vergangenen Woche (zum Beispiel MONDAY) eingegeben werden. Ebenfalls erlaubt ist TODAY oder YESTERDAY (für »heute« oder »gestern«).
UPTO	<Datum> listet nur Dateien auf, die am bezeichneten Datum oder früher bearbeitet wurden. Sonst wie SINCE.
QUICK	zeigt nur die Namen der Dateien und Verzeichnisse an. Alle anderen, die Anzeige manipulierenden Optionen werden unterdrückt.
BLOCK	zeigt die Dateigröße in Blöcken und nicht in Bytes an.
NOHEAD	weglassen der Kopf- und Fußzeile des LIST-Befehls.
FILES	zeigt nur die Dateinamen in dem Vezeichnis an, Unterverzeichnisse werden nicht ausgegeben.
DIRS	nur Verzeichnisnamen werden aufgelistet.
LFORMAT	spezifiziert ein besonderes Ausgabeformat für den Befehl LIST und läßt sich zum Erstellen von Script-Dateien nutzen, wenn die Ausgabe mit »>« in eine Datei umgeleitet wird. Die Optionen NODATES, QUICK und NOHEAD werden automatisch eingeschaltet.

Das Ausgabeformat wird mit

```
LIST ... LFORMAT="Formatzeichenkette"
```

festgelegt. Damit können Texte und die Ausgabe von LIST kombiniert werden. Informationen aus der Ausgabe von LIST lassen sich durch Einsetzen von »%S« in die Formatzeichenkette in die Ausgabe einfügen. Welcher Teil der Ausgabe von LIST in die Ausgabe eingesetzt wird, hängt von der Anzahl der »%S«-Anweisungen ab:

Anzahl von %S	Ausgabe
1	Nur Dateiname
2	Pfad, Dateiname
3	Pfad, Dateiname, Pfad
4	Pfad, Dateiname,Pfad, Dateiname

Beispiel:

```
LIST #? TO t:pro.bat FILES LFORMAT="protect %S -d"
```

erzeugt die Datei »pro.bat« auf dem logischen Gerät »T«:.

```
protect life.c -d
protect lmkfile -d
protect prof.out -d
protect 3dl -d
protect 3dl.c -d
protect 3dl.h -d
protect 3dl.lnk -d
protect 3dl.map -d
protect 3dl.o -d
```

Mit dem Befehl

```
EXECUTE t:pro.bat
```

kann man nun das Löschschutzflag der oben genannten Dateien löschen, das heißt die Dateien vor dem Löschen schützen.

```
LIST USR:Quelltexte SUB Harribald
```

listet alle Dateien und Verzeichnisse aus dem Verzeichnis »USR:Quelltexte« die die Zeichenfolge »Quelltexte« in ihrem Namen haben. Beachten Sie, daß die Eingabe:

```
LIST SUB
```

die Fehlermeldung »args no good for key« verursacht, auch, wenn es das Verzeichnis SUB im aktuellen Verzeichnis gibt. Dessen Inhalt erhalten Sie mit der Eingabe:

```
LIST "SUB"
```

Die Eingabe von

```
LIST arbeit P neu#?(x|y)
```

listet alle Dateien, die mit »neu« beginnen und mit »x« oder »y enden.

LOADWB

Format: LOADWB [DELAY] [-DEBUG]

Muster: LOADWB "DELAY/S,-DEBUG/S"

Zweck: Starten der Workbench.

Pfad: C:LOADWB

Beschreibung:

Der Befehl LOADWB wird verwendet, um die Workbench-Benutzeroberfläche zu starten. Dies erfolgt meist automatisch beim Reboot des Systems, indem LOADWB in der Datei »S:Startup-Sequence« eingetragen wird. In der original Workbench steht er bereits in der Startup-Sequence.

Die Workbench überprüft, welcher Suchpfad für DOS-Befehle zu dem Zeitpunkt mit PATH gesetzt ist, wenn die Workbench gestartet wird, und übergibt ihn an jeden SHELL-Prozeß, der von der Workbench aus gestartet wird.

Die Option DELAY veranlaßt LOADWB vor der Beendigung drei Sekunden zu warten, damit Diskettenaktivitäten abgeschlossen werden können, bevor die Ausführung der Script-Datei fortgesetzt wird.

DEBUG aktiviert ein sonst verborgenes Workbench-Menü, das die Menüs Debug und Flush-Libs enthält. Debug startet den ROM-residenten Debugger ROMWACK, der seine Ausgaben über die serielle Schnittstelle mit 9600 Baud auf ein angeschlossenes Terminal leitet.

Flush-Libs veranlaßt, daß im Rechner aufgeräumt wird, und alle unbenutzten diskresidenten Teile des Betriebssystems aus dem Speicher gelöscht werden, wenn sie nicht von irgendwelchen Prozessen benutzt werden. Alle Bibliotheken, Gerätetreiber, Zeichensätze und andere Resourcen, die sich resident im Speicher befinden, nachdem sie von Diskette oder Festplatte geladen und im System installiert wurden, bleiben normalerweise so lange auch unbenutzt im Speicher des Rechners stehen, bis das Betriebssystem durch akuten Speicherplatzmangel anfängt sie zu löschen, um wieder etwas mehr Speicher für Programme frei zu haben.

Die beiden Optionen DELAY und DEBUG können nicht gleichzeitig verwendet werden.

LOCK

Format: LOCK <Laufwerk>: [ON|OFF] [<Paßwort>]

Muster: LOCK "LAUFWERK/A,ON/S,OFF/S,PASSWORT"

Zweck: Aktivieren des Schreibschutzes einer Festplatte bei Verwendung des Fast-Filing-System.

Pfad: C:LOCK

Beschreibung:

Der Befehl

```
LOCK <Festplatte>: ON
```

aktiviert den »physikalischen« Schreibschutz einer Festplattenpartition, die das Fast-Filing-System benutzt. LOCK simuliert also den Schreibschutzschieber von 3,5-Zoll-Disketten.

```
LOCK <Laufwerk>: OFF
```

Schaltet den Schreibschutz ab. OFF kann durch einen Fehler im Befehl LOCK auch weggelassen werden. (Diese kleine Unsauberkeit wird hoffentlich bis zur Version 1.4 des Betriebssystems beseitigt.) Wahlweise kann ein Paßwort von 4 Zeichen Länge angefügt werden, das zum Aufheben des Schreibschutzes ebenfalls angegeben werden muß, da sonst der Befehl zurückgewiesen wird.

LOCK wirkt bis zum nächsten Reboot.

MAKEDIR

Format: MAKEDIR <Verzeichnis>

Muster: MAKEDIR "Verzeichnis/A"

Zweck: Anlegen eines Verzeichnisses.

Beschreibung:

MAKEDIR erzeugt ein neues Verzeichnis mit dem angegebenen Namen als Unterverzeichnis des aktuellen Verzeichnisses, wenn nicht der Pfad angegeben wurde, der zum neu anzulegenden Verzeichnis hinführen wird. Der Befehl erzeugt nur das verlangte Unterverzeichnis, aber keine dazu übergeordneten Verzeichnisse. Daher müssen alle höheren Verzeichnisse im Pfad zu dem neu zu erzeugenden Verzeichnis bereits bestehen.

Beispiel(e):

```
MAKEDIR test
```

erzeugt ein Verzeichnis mit Namen »test« im aktuellen Verzeichnis.

```
MAKEDIR DF1:xyz
```

erzeugt das Verzeichnis »xyz« im Haupt-Verzeichnis der Diskette in Laufwerk DF1:.

```
MAKEDIR DF1:XYZ/ABC
```

erzeugt das Verzeichnis »ABC« im Verzeichnis »XYZ« auf der Diskette im Laufwerk DF1:. Das Verzeichnis »XYZ« muß dazu bereits bestehen.

Siehe auch: DELETE

MERGEMEM

Format: MERGEMEM

Muster: MERGEMEM

Zweck: Physikalisch getrennte Speicherblöcke auf verschiedenen Speichererweiterungskarten zu einem Block zusammenfassen.

Pfad: SYSTEM:MERGEMEM

Beschreibung:

Wenn in einem Amiga mehrere Speichererweiterungsplatinen vorhanden sind, so trägt die Autokonfig-Routine beim Systemstart jede einzelne Speichererweiterung in die Systemspeicherliste als separaten Eintrag ein (siehe auch FASTMEMFIRST).

MERGEMEM versucht nun diese einzelnen Speicherblöcke zu einem großen Block zu verbinden, um es Progammen zu ermöglichen, auch sehr große Speicherblöcke vom Betriebssystem am Stück anzufordern.

War es nicht möglich die verschiedenen Speicherblöcke zusammenzufassen, so wird die endgültige RAM-Konfiguration angezeigt und gemeldet, daß ein weiteres Zusammenfassen nicht möglich war.

MORE

Format: MORE <Dateiname>

Muster: MORE "DATEINAME/A"

Zweck: Komfortables Anzeigen einer Textdatei

Pfad: WORKBENCH1.3D:UTILITIES/MORE

Beschreibung:

MORE zeigt ASCII-Textdateien komfortabel an. Wenn das Anzeigefenster mit Text gefüllt ist, hält MORE die Ausgabe an, und gibt den prozentualen Anteil des Gesamttextes an, der bisher angezeigt wurde. Um die Ausgabe nun fortzuführen sind folgende Eingaben erlaubt:

[Leertaste]	Zeigt die nächste Seite
[Return]	Zeigt die nächste Zeile
[Backspace]	Zeigt die vorhergehende Seite *
<	Zeigt die erste Seite *
>	Zeigt die letzte Seite *
%n	Zeigt den Inhalt der Datei bei zirka n Prozent des Textes *
[Ctrl]+[L]	Baut den Inhalt des Fensters neu auf
/<Zeichenfolge>	MORE sucht nach der Zeichenfolge, die auf den "/" folgend angegeben wurde. Groß- und Kleinschreibung wird berücksichtigt (Identität)!
.<Zeichenfolge>	MORE sucht nach der Zeichenfolge, die angegeben wurde. Groß- und Kleinschreibung wird nicht berücksichtigt.

<n>	Das nächste Auftreten der Zeichenfolge wird gesucht.
<h>	Hilfe; zeigt eine Liste der Befehle und ihrer Bedeutung an
<q>	(oder [Ctrl]+[C]) Abbruch von MORE
[Shift]+[E]	Lädt den Text zum Bearbeiten in den Editor, der in der Umgebungsvariable ENV:EDITOR festgelegt ist (siehe auch: SETENV).

Wird die PIPE: als Eingabe für MORE verwendet, sind die mit einem Sternchen (*) gekennzeichneten Befehle wirkungslos, da die Länge einer von der PIPE: gelesenen Datei nicht bekannt ist.

Wenn das Ende der Datei erreicht wurde, erscheint am unteren Fensterrand die Anzeige "--- Less ---", ein weiterer Druck auf [Leertaste] oder [Return] beendet nun MORE.

Himweis: Wird [Esc] gedrückt, wird MORE unterbrochen. Nach Drücken der Tasten [q] oder [h] kann die Ausgabe fortgesetzt werden.

MOUNT

Format: MOUNT <Gerät> [FROM <Datei>]

Schablone: MOUNT "GERÄT/A,FROM/K"

Zweck: Anmeldung eines neuen Amiga-DOS-Geräts.

Pfad: C:MOUNT

Beschreibung:

MOUNT teilt dem Amiga-DOS mit, daß ein zusätzliches Gerät vom Dateisystem verwendet werden soll. MOUNT sucht in der Datei DEVS:MountList (oder der Datei, die wahlweise mittels FROM angegeben wurde) nach den Parametern des Geräts, das angemeldet wurde.

Siehe auch: Mountlist Kapitel 5

NEWCLI

Format: NEWCLI [<Fenster>] [FROM <Script-Datei>]

Muster: NEWCLI "FENSTER, FROM/K"

Zweck: Ein neues CLI-Fenster öffnen.

Pfad: C:NEWCLI

Beschreibung:

Amiga-DOS öffnet ein neues CLI-Fenster. Das neue Fenster wird automatisch aktiviert. Bevor der Benutzer das CLI-Fenster interaktiv nutzen kann, wird automatisch die Script-Datei S:CLI-Startup ausgeführt, falls nicht mit FROM eine andere Befehlsdatei benannt wurde.

Wird NEWCLI ohne Argument gestartet, erzeugt Amiga-DOS ein Fenster in der Standardgröße und an der Standardstelle, die im Kickstart-ROM voreingestellt ist. Um die Größe des Fensters zu verändern, bringen Sie den Mauszeiger an die untere rechte Ecke, auf das grafische Symbol zum Ändern der Fenstergröße, betätigen den linken Mausknopf und halten ihn gedrückt. Darauf können Sie das Fenster mit der Maus vergrößern oder verkleinern. Um die Fensterposition zu ändern, bewegen Sie den Mauszeiger zur Kopfleiste, betätigen wiederum den linken Mausknopf und bringen die Maus an die Stelle, an der das Fenster gewünscht wird. Um das CLI-Fenster den Benutzerwünschen anzupassen, kann man jedoch die genaue Position, Größe und einen neuen Namen eingeben. Die Syntax, um die Eigenschaften eines neuen Fensters zu beschreiben lautet:

```
CON:x/y/Breite/Höhe/Name
```

wobei CON: für ein Fenster steht, X und Y für die Koordinaten der linken oberen Ecke des Fensters, Breite und Höhe für die Fenstergröße relativ zur linken oberen Ecke. Name ist die Zeichenkette, die in der Titelzeile des CLI-Fenster angezeigt werden soll. Der Name muß nicht angegeben werden und kann leer bleiben, der letzte Schrägstrich (/) muß dagegen immer gesetzt werden. Alle Größenangaben beziehen sich auf Bildpunkte auf dem Bildschirm.

Beispiel(e):

```
NEWCLI
```

erzeugt ein neues CLI und macht es zum aktuellen.

```
NEWCLI CON:10/30/300/100/MEINCLI
```

erzeugt ein neues CLI. Das Fenster beginnt bei Position 10,30 und ist 300 * 100 Pixel groß. Es trägt den Namen MEINCLI.

```
NEWCLI "CON:10/30/300/100/MEIN CLI"
```

erzeugt das gleiche Fenster, erlaubt jedoch durch die Anführungszeichen (") die die Verwendung von Leerzeichen innerhalb des Namens.

Siehe auch: ENDCLI, RUN

NEWSHELL

Format: NEWSHELL [<Fenster>] [FROM <ScriptDatei>]

Muster: NEWSHELL "WINDOW,FROM/K"

Zweck: Starten einer neuen interaktiven SHELL.

Pfad: C:NEWSHELL

Beschreibung:

Der Befehl NEWSHELL startet eine neue, interaktive SHELL und verwendet den Console-Handler NEWCON:, falls dieser mit

```
MOUNT NEWCON:
```

installiert wurde, sonst wird das alte CON: verwendet, das den Bedienungskomfort der SHELL einschränkt. Außerdem muß »L:SHELL-SEG« resident im Speicher installiert werden. (Dies geschieht in der Standard-Startup-Sequenz der Workbench1.3-Diskette.) Falls »L:SHELL-SEG« nicht resident geladen wurde, öffnet NEWSHELL ein neues CLI-Fenster.

NEWSHELL führt beim Aufruf die Script-Datei »S:SHELL-STARTUP« aus, sofern nicht mittels FROM ein anderer Dateiname angegeben wird.

siehe auch: NEWCLI, ENDCLI

NOFASTMEM

Format: NOFASTMEM

Muster: NOFASTMEM

Zweck: Abschalten aller RAM-Erweiterungen bis auf das CHIP-RAM

Pfad: SYSTEM:NOFASTMEM

Beschreibung:

Einige sehr schlecht programmierte Programme, die (hoffentlich) aus den Anfangstagen des Amiga stammen, laufen nur auf Amigas, die nur CHIP-RAM besitzen, das heißt auf Amigas ohne jede Speichererweiterung über 512 Kbyte (neue Fat-Agnus 1 Mbyte) hinaus.

NOFASTMEM blendet deshalb alle Speichererweiterungen aus, damit diese Software laufen kann. (Manchmal ist es aber lohnender, die Speichererweiterung nicht abzuschalten und das Programm dafür nicht zu benutzen).

NOFASTMEM arbeitet als Schalter, nach dem ersten Aufruf wird der Speicher ausgeblendet, nach dem zweiten Aufruf wird der Speicher wieder eingeblendet. Falls ein Programm schon im FAST-RAM ausgeführt wird, wenn das RAM ausgeblendet wird, kann es eventuell in manchen Fällen zu Abstürzen des Rechners kommen.

PATH

Format: PATH [SHOW] | [ADD <Verzeichnisname>* [RESET] [QUIET]

Muster: PATH,,,,,,,,,ADD/S,SHOW/S,RESET/S,QUIET/S

Zweck: Verzeichnisse, in denen Amiga-DOS nach einem Kommando sucht, werden hinzugefügt, geändert oder ausgegeben.

Pfad: C:PATH

Beschreibung:

Nach Voreinstellung sucht Amiga-DOS zunächst in dem aktuellen Verzeichnis, dann im Verzeichnis SYS:C nach einem auszuführenden Befehl. Mit PATH kann nun dieser Suchpfad verändert oder erweitert werden. Geben Sie

```
PATH RESET
```

ohne Verzeichnis-Namen ein, dann wird diese Grundeinstellung wiederhergestellt. Wird eine Reihe von Verzeichnisnamen angegeben, so wird der momentane Suchpfad durch diese Verzeichnisse ersetzt, die dann in ihrer angegebenen Reihenfolge durchsucht werden, um eine eingegebene Anweisung zu finden.

Mit der Option ADD werden die angegebenen Verzeichnisse an den bestehenden Suchpfad angefügt. Sie können bis zu zehn Verzeichnisse mit einem einzigen PATH ADD hinzufügen. Die Namen müssen durch mindestens ein Leerzeichen voneinander getrennt sein. Das aktuelle Verzeichnis wird immer als erstes durchsucht, egal welcher Suchpfad gesetzt wurde.

PATH SHOW oder PATH ohne Optionen zeigt den aktuellen Suchpfad an. Die Schlüsselwörter RESET, ADD und SHOW können am Anfang oder am Ende der Parameterzeile angegeben werden.

Beispiel(e):

```
PATH SHOW (oder PATH)
```

Zeigt die Verzeichnisse an, in denen Amiga-DOS Kommandos sucht. Ein typischer Suchpfad lautet:

```
Current directory (aktuelles Verzeichnis)
WB1.3:s
WB1.3:Bin
WB1.3:Utilities
WB1.3:System
WB1.3:Prefs
C:
```

Geben Sie danach den folgenden Befehl ein, wird Amiga-DOS nun auch im Verzeichnis »SYS:newcommands« nach Kommandos suchen:

```
PATH ADD SYS:newcommands
```

PATH zeigt jetzt folgenden Suchpfad an:

```
Current directory (aktuelles Verzeichnis)
WB1.3:s
WB1.3:Bin
WB1.3:Utilities
WB1.3:System
WB1.3:Prefs
SYS:newcommands
C:
```

In der Startup-Sequenz kann jeder Anwender den Suchpfad so setzen, daß automatisch beim Booten unter Berücksichtigung der persönlichen Umstände alle oft benutzten Programme direkt im Suchpfad liegen, und nur durch Angabe ihres Namens gestartet werden können.

```
Path s: sys:bin sys:utilities sys:system sys:prefs
```

ist ein Befehl, der alle wichtigen Systembefehle zugänglich macht. Schauen Sie doch einmal in Ihre Startup-Sequenz. Der Befehl PATH wird dort sicherlich auftauchen!

Die Option QUIET bewirkt, das PATH bei der Option SHOW nicht nach vorhandenen Disketten oder logischen Geräten fragt. Wurde ein Gerät nicht durch MOUNT angemeldet, so wird nicht der Pfad des Verzeichnisses angezeigt, das in den Suchpfad aufgenommen werden sollte, sondern lediglich der Name des Geräts.

PREFERENCES

Format: PREFERENCES [POINTER|PRINTER|SERIAL]

Muster: PREFERENCES "POINTER | PRINTER | SERIAL"

Zweck: Verändern der Systemeinstellungen

Pfad: PREFS/PREFERENCES

Beschreibung:

Mit dem Werkzeug Preferences lassen sich die Systemeinstellungen verändern.

RESET ALL holt die Einstellungen der original Workbench zurück.

LAST SAVED lädt aus dem Verzeichnis DEVS: die letzte abgespeicherte Version der Systemkonfiguration, die in der Datei DEVS:System-Configuration gespeichert wird.

Datum und Uhrzeit können gestellt werden, und werden mit SAVE auch in die batteriegepufferte Echtzeituhr übernommen, die im Amiga 2000 und auf der internen Speichererweiterung für den Amiga 500 vorhanden ist.

Mit TEXT 60/80 kann zwischen den zwei im ROM befindlichen Zeichensätzen Topaz 8 und Topaz 9 gewählt werden. Mit Fast-Fonts (FF) können andere nicht proportionale Zeichensätze als Systemfont benutzt werden.

Die Schieberegler im unteren Teil des PREFERENCES-Fenster dienen zum Einstellen der vier Workbenchfarben.

KEY REPEAT DELAY dient zum Festlegen des Zeitraums, den eine Taste niedergedrückt werden muß, damit die Wiederholfunktion aktiviert wird.

KEY REPEAT SPEED legt fest, wie groß die Frequenz der Tastenwiederholung ist.

WORKBENCH INTERLACE legt fest, ob die Workbench im Interlace-Modus oder im normalen Hires-Modus betrieben werden soll. Der Interlace-Modus stellt die doppelte Anzahl Zeilen wie der Hires-Modus auf dem Bildschirm dar, setzt aber die Bildwiederholfrequenz von 50 Hz auf 25 Hz herab, was zu einem starken Flimmern des Bildschirms führt, wenn nicht ein Flicker-Fixer (Grafikkarte von der Firma Micro-Way für den Amiga 2000 B, die je zwei interlace Halbbilder zwischenspeichert und als Vollbild mit 50 Hz Bildwiederholfrequenz auf einen multisynchronen Monitor ausgibt) oder ein sehr lang nachleuchtender Monitor verwendet wird.

Die beiden mit Icons der Maus zugeordneten Einsteller sind für die »Übersetzung« der Maus und den Zeitraum, innerhalb dessen ein Doppelklick als Doppelklick erkannt wird, zuständig.

Mit dem großen rechteckigen Einsteller in der Mitte kann die Workbench auf dem Bildschirm zentriert werden.

Das Feld CHANGE SERIAL aktiviert ein weiteres Fenster, daß die Einstellung der Parameter für die serielle Schnittstelle zuläßt.

EDIT POINTER startet den Editor für den Mauszeiger.

CHANGE PRINTER startet das Druckerauswahlmenü.

CANCEL verwirft alle getätigten Einstellungen und verläßt Preferences.

USE übernimmt die Einstellungen zum Arbeiten, speichert sie aber nicht in der Datei »DEVS:System-Configuration« ab.

SAVE übernimmt die Einstellungen und speichert sie ab.

Die Kommandozeilen Optionen SERIAL, PRINTER und POINTER rufen das jeweilige Einstellmenü direkt auf, Preferenzes arbeitet dann ganz normal, und kann über CANCEL, USE oder SAVE verlassen werden.

Im Einstellmenü für den Drucker können die Schnittstelle, der Druckertreiber, das Papierformat und Parameter für den Textdruck festgelegt werden. Ferner sind zwei weitere Menüs anwählbar, die den Grafikdruck steuern.

Im Menü GRAPHIC 1 kann mit THRESHOLD der Schwellwert eingestellt werden, ab dem ein Farbton bei der Grauwertumsetzung als schwarz gedruckt wird, je kleiner der Wert, um so dunkler muß die Farbe sein.

ASPECT wählt Hoch- oder Querformat, IMAGE positive oder negative Darstellung des Ausdrucks.

Unter SHADE wird festgelegt, ob der Ausdruck schwarz/weiß (BLACK AND WHITE), farbig (COLOR) oder in Graustufen (GRAY SCALE) erfolgen soll. Der Punkt (GRAY SCALE2) dient zum Ausdruck einer Bildschirmseite des sogenannten Hedley-Monitors. Der Hedley-Monitor ist ein hochauflösender Graustufenmonitor, der eine Auflösung von 1008 mal 1024 Punkten in der europäischen Version darstellen kann. (Er ist bei der Erstellung des Manuskripts im ersten Quartal 1989 noch nicht im Handel erhältlich). Dieser Monitor benötigt wegen seiner hohen Auflösung spezielle Grafiksystemsoftware, die mit ihm zusammen ausgeliefert werden wird.

Im Menü GRAPHIC 2 stehen folgende Optionen zur Auswahl:

SMOOTHING aktiviert die Antialiasing-Funktion, die versucht, Treppchen beim Ausdruck von diagonalen Linien zu glätten. Beim Ausdruck von Bildschirmgrafiken steht man vor zwei Problemen. Erstens müssen die Bildschirmfarben in Graumuster umgesetzt werden, oder auf einem farbfähigen Drucker aus den Elementarfarben die einzelnen Töne gemischt werden. Zweitens muß die recht niedrige Auflösung einer Bildschirmgrafik durch mathematische Tricks an die wesentlich höhere Auflösung des Druckers angepaßt werden.

Bildschirmpunkte werden also aus mehreren Punkten auf dem Papier zusammengesetzt. Das führt bei diagonalen Linien oder Kreisen, allgemein an Kanten mit großem Hell-Dunkelkontrast zu einer sichtbaren »Körnung« oder Treppchen im Ausdruck.

Antialiasing ist ein Verfahren, um solche Ränder etwas zu glätten, »weicher« zu machen. SMOOTHING funktioniert nicht im Zusammenhang mit dem Floyd-Steinberg (F–S) Schattierungsalgorithmus, der Bildschirmpunkte nicht mit regelmäßigen Rastern füllt, sondern mit der Helligkeit des Bildschirmpunktes entsprechenden Zufallsmustern, was SMOOTHING völlig durcheinanderbringen würde. Wird SMOOTHING bei aktivierter F–S Schattierung angewählt, wird die Schattierung automatisch zu ORDERED umgeschaltet.

LEFT OFFSET gibt die horizontale Verschiebung des gedruckten Bildes an. Diese Option erlaubt die Einstellung der Breite des linken Randes. Die Verschiebung wird in Vielfachen von 1/10 Inch (Zoll) angegeben. CENTER, deaktiviert die Verschiebung nach links. Standardeinstellung ist 0,0 Inch.

CENTER zentriert das gedruckte Bild horizontal. Diese Option hebt den für LEFT OFFSET festgelegten Wert auf.

DENSITY [1 bis 7] wählt die Druckdichte der Grafik. Je niedriger die Druckdichte, um so schneller der Druck, aber desto niedriger die Feinheit des Ausdrucks. Die Beschreibung der Dichten, die von den einzelnen Druckern unterstützt werden, ist dem Anhang Druckertreiber zu entnehmen.

COLOR CORRECT R/G/B versucht, die Farben des Drucks besser an die der Bildschirmdarstellung anzugleichen. Diese Option versucht je nach Auswahl, alle Farbtöne von Rot (R), Grün (G) oder Blau (B) im Ausdruck mit denen auf dem Bildschirm in Übereinstimmung zu bringen. Ohne Farbkorrektur kann ein Farbdrucker bis zu 4096 Farbtöne drucken. Wird jedoch die Farbkorrektur gewählt, wird zugunsten der Farbtreue auf die Maximalzahl von verschiedenen Farbtönen verzichtet. Pro angewählter Grundfarbe gehen 308 Schattierungen verloren.

Unter DITHERING kann der Schattierungsalgorithmus ausgewählt werden. Floyd-Steinberg verwendet, wie oben schon erwähnt, mit einem Zufallsmuster gefüllte Grauflächen, Halftone stellt jeden Bildschirmpunkt mittels konzentrischer Graumuster dar, und Ordered verwendet regelmäßige rechteckige Grauraster.

SCALING legt fest, ob die Bildpunkte in gerundeten Abständen gesetzt werden dürfen (FRACTION), oder jeder Bildpunkt in eine konstante Anzahl Nadelpunkte umgesetzt wird (INTEGER).

WIDTH LIMIT begrenzt die Breite des gedruckten Bildes durch eine Breitenangabe in entweder 1/10 Inch (Zoll), Bildpunkt oder mit einem Multiplikationsfaktor und beschreibt damit eine Einstellung aus dem Punkt LIMITS näher.

HIGHT LIMIT bestimmt wie WIDTH LIMIT die Höhe des Bildes.

Mit LIMITS wird bestimmt, wie die Werte für die Begrenzung des Bildausschnitts ausgewertet werden.

IGNORE bewirkt, daß die Begrenzungen ignoriert werden. Die Größe des gedruckten Bildes wird durch das Anwendungsprogramm wie folgt festgelegt:

Breite = (rechter Rand – linker Rand + 1) / Zeichen pro Inch
Höhe = Zeilen pro Seite/Zeilen pro Inch

BOUNDED: Die Bildgröße wird durch die Werte eingegrenzt, die bei WIDTH LIMIT und HIGHT LIMIT angegeben werden. Hiermit kann jeweils die Größe eines Bildes unabhängig von der Einstellung für Textausdrucke genau in 1/10 Zoll angegeben werden.

ABSOLUTE: Die Begrenzungen der Breite und der Höhe werden als Absolutwerte aufgefaßt. Das Bild wird genau in der angegebenen Größe in 1/10 Zoll gedruckt, ohne daß das Verhältnis von Höhe zu Breite automatisch ausgeglichen wird, so daß die Bilder je nach Einstellung stark verzerrt ausgedruckt werden können.

PIXELS: Die Werte der Breiten- und Höhenbegrenzung werden als Anzahl der Bildpunkte angegeben.

MULTIPLY: Die Werte der Breiten- und Höhenbegrenzung werden als Multiplikator für die Breite und Höhe des Ausgangsbildes in Pixeln interpretiert.

Verwenden Sie eine Festplatte, müssen aber von Diskette booten, ist es sinnvoll die Datei »DEVS:System-Configuration« vom DEVS:-Ordner, der sich auf der Festplatte befindet, auf die Bootdiskette zu kopieren (siehe COPY), da Preferences veränderte Systemkonfigurationen in das logische Gerät DEVS: schreibt, das meist durch einen entsprechenden ASSIGN-Befehl in der Startup-Sequence auf ein Verzeichnis auf der Festplatte weist. Das Utility COPYPREFS, eine Script-Datei, die mit ICONX von der Workbench ausgeführt wird, nimmt ihnen diese Arbeit ab.

Um einen individuellen Druckertreiber in das DEVS:-Verzeichnis zu kopieren kann das Utility INSTALLPRINTER aus dem Verzeichnis »WORKBENCH1.3D:UTILITIES« benutzt werden oder auf der SHELL

```
COPY Extras1.3D:devs/printers/<Treibername> to devs:printers
```

eingegeben werden.

PRINTFILES

Format: PRINTFILES [–f] <Dateiname> [[–f] <Dateiname>] [[–f] <Dateiname>] . . .

Muster: PRINTFILES "–F, DATEINAME, ..."

Zweck: Ausdruck von Textdateien.

Pfad: WORKBENCH1.3D:UTILITIES/PRINTFILES

Beschreibung:

PRINTFILES dient zum Ausdruck von Textdateien über den mit PREFERENCES festgelegten Drucker. Die Option »–f« schickt am Ende jeder Datei einen Form Feed an den Drucker, wirft also das vorige Blatt aus und beginnt eine neue Seite.

Wird PRINTFILES von der Workbench aus gestartet, müssen zuerst die Dateien, die ausgedruckt werden sollen, einmal angeklickt werden (bei mehreren die zweite und jede weitere bei niedergedrückter [Shift]-Taste) und danach bei gedrückter Shift-Taste PRINTFILES mit einem Doppelklick gestartet werden. Mit Hilfe des Workbench-Menüpunkts INFO kann PRINTFILES im Feld TOOLTYPES seines Icons die Option »–f« übergeben werden, indem dort

```
"FLAGS=FORMFEED"
eingegeben wird.
```

PROMPT

Format: PROMPT <prompt>

Muster: PROMPT "PROMPT"

Zweck: Verändern des Prompt.

Pfad: C:PROMPT

Beschreibung:

Mit PROMPT kann das SHELL-Eingabebereitschaftszeichen (Prompt) in der aktuellen SHELL verändert werden. Geben Sie keine Parameter an, setzt Amiga-DOS das Eingabebereitschaftszeichen auf den Standard-Prompt »n>« zurück. Anderenfalls wird das bisherige Prompt durch die eingegebene Zeichenkette ersetzt. PROMPT akzeptiert auch zwei spezielle Anweisungen, »%N« und »%S«.

%N Zeigt nur die Nummer des Befehlsprozessors an.
%S Zeigt das aktuelle Verzeichnis an.

Beispiel(e):

```
PROMPT
```

setzt das Prompt zum normalen Zeichen zurück.

```
PROMPT "%N.%S>"
```

zeigt die laufende Nummer des SHELL-Prozesses und davon mit einem Punkt getrennt das aktuelle Verzeichnis an, in dem sich der Benutzer befindet.

PROTECT

Format: PROTECT [FILE] <Dateiname> [FLAGS +|– <status>][ADD] [SUB]

Muster: PROTECT "FILE, FLAGS/K, ADD/S, SUB/S"

Zweck: Ändern der Schutzflags einer Datei

Beschreibung:

Mit den Schlüsselworten ADD oder »+« können Schutzflags gesetzt, mit SUB oder »–« gelöscht werden.

Es gibt folgende Schutzflags:

»r« für lesen (read). Das Lesen dieser Datei ist gestattet. Dieses Schutzflag wird vom alten File-System nicht unterstützt, unter Fast-Filing-System ist der Nutzen auch sehr zweifelhaft. (Hinterlassenschaft des Multiusersystems TRIPOS)

»w« für schreiben (write). Auf diese Datei dürfen Daten geschrieben werden. Wird wie »r« nur vom Fast-Filing-System unterstützt.

»e« für ausführbar (execute), wird vom DOS nicht benutzt, die SHELL wertet das Flag aus, um festzustellen, ob die Datei ein ausführbares Programm ist.

»d« für löschbar (delete). Ist das Flag gelöscht, kann die Datei nicht mit DELETE gelöscht werden.

»s« für eine Script-Datei, die auch ohne EXECUTE durch alleinige Angabe des Namens der Script-Datei, ähnlich wie ein DOS-Befehl, ausführbar ist (script).

»p« (pure) für Programme, die reentrant (wiedereintrittsfähig) und multientrant (mehrfach gleichzeitig benutzbar) sind, und mit dem Befehl RESIDENT resident gemacht werden können. Diese Programme können genau einmal in den Speicher geladen werden und verbleiben dort auch nach der Ausführung, was bei erneuter Benutzung ein neuerliches Laden unnötig macht. Sie können beliebig oft (gleichzeitig und nacheinander) benutzt werden.

»a« für archiviert; Das Archivflag einer Datei wird stets gelöscht, wenn auf eine Datei geschrieben wurde. Das Archivflag eines Verzeichnisses wird gelöscht, wenn auf eine Datei in diesem Verzeichnis geschrieben wird. Das ermöglicht einem Archivierungs-(Backup-)Programm, eine Diskette oder Festplattenpartition nach solchen Dateien zu durchsuchen, die seit dem letzten Aufruf des Programms verändert wurden, um dann gezielt nur von diesen Dateien eine Sicherheitskopie anzufertigen.

Beispiel(e):

```
PROTECT help-script FLAGS +s
```

erlaubt das »help-script« ohne EXECUTE ausgeführt werden kann.

```
PROTECT programm2 +rwd
```

erlaubt nur die Zugriffe lesen, schreiben und löschen auf programm2.

Ist der Zugriff »schreiben« nicht erlaubt, kann die Datei zum Beispiel nicht mit COPY überschrieben werden, Zugriffe mit dem Editor sind aber weiterhin möglich. Ebenso kann der Inhalt einer Datei mit TYPE auch dann noch ausgegeben werden, wenn der Zugriff »lesen« nicht erlaubt ist.

Beispiel(e):

Das Schutzflag e (ausführbar) der Datei "Harribald" im Verzeichnis USR: wird gelöscht.

```
PROTECT USR:Harribald -e
```

Siehe auch: LIST

READ

Format: READ [TO]<name> [SERIAL]

Muster: READ "TO/A, SERIAL/S"

Zweck: READ liest Daten aus der parallelen oder der seriellen Schnittstelle und speichert sie in einer Datei.

Pfad: Wie auch DOWNLOAD ist READ nicht auf der normal erhältlichen Workbench vorhanden, und wird für Entwickler getrennt mit dem Crossdevelopment-Package zur Verfügung gestellt.

Beschreibung:

Der Befehl READ erwartet und übernimmt einen Strom von hexadezimalen Zeichen von der parallelen Schnittstelle. Geben Sie den Zusatz SERIAL ein, liest READ statt dessen von der seriellen Schnittstelle. Für jedes Nibble (vier Bit) der zu empfangenden Datei muß eine ASCII-Hexziffer (READ erkennt nur die ASCII-Zahlen 0 bis 9 und die Großbuchstaben A bis F) gesendet werden, jedes Hexziffernpaar wird als ein Byte im Speicher abgelegt. Der gesendete Datenstrom muß also eine gerade Anzal von Hexziffern und ein abschließendes »Q« beinhalten.

READ behandelt »Q« als Endmarkierung des Zeichenstroms. READ übergeht Leerzeichen, neue Zeilen und Tabulatoren. Ist der Datenfluß beendet, schreibt READ die Bytes aus dem Speicher in die bezeichnete Datei. (Sie können sowohl Text- als auch binäre Dateien übertragen.)

Vorsicht, wenn Sie denselben Dateinamen zweimal verwenden. READ überschreibt vorhandene Dateien ohne Warnung. Wählen Sie die Baud-Rate nicht zu hoch. Die Gefahr des Datenverlustes ist dann zu groß.

Beispiel(e):

```
READ TO DF0:neu
```

liest Daten vom parallelen Port und schreibt sie in die Datei »neu« auf die Diskette in Laufwerk DF0:.

```
READ neu SERIAL
```

liest Daten vom seriellen Port und schreibt sie in die Datei »neu« in das aktuelle Verzeichnis.

RELABEL

Format: RELABEL [DRIVE]<drive> [NAME]<name>

Muster: RELABEL "DRIVE/A, NAME/A"

Zweck: Ändern eines Diskettennamens.

Pfad: C:RELABEL

Beschreibung:

RELABEL ändert den Namen einer Diskette in den hinter NAME bezeichneten neuen Namen. Diskettennamen werden üblicherweise beim Formatieren einer Diskette festgesetzt.

Beispiel:

```
RELABEL DF1: "Neuer Name"
```

ändert den Namen der Diskette im Laufwerk DF1: zu »Neuer Name«.

Siehe auch: FORMAT

REMRAD

Format: REMRAD

Schablone: REMRAD

Zweck: Entfernen der resetfesten RAM-Disk

Pfad: C:REMRAD

Beschreibung:

Soll die resetfeste RAM-Disk gelöscht und aus dem Speicher des Amiga entfernt werden, ohne den Computer abzuschalten, hilft REMRAD. REMRAD weist die resetfeste RAM-Disk an, alle darin enthaltenen Dateien zu löschen. Auch ihr Speicherbedarf wird auf ein Minimum verringert, so daß nur noch sehr wenig Speicher verbraucht wird. Beim nächsten Reboot des Amiga wird dann die resetfeste RAM-Disk aus dem System entfernt.

RENAME

Format: RENAME [FROM]<name> [TO|AS]<name>

Muster: RENAME "FROM/A, TO=AS/A"

Zweck: Eine Datei oder ein Verzeichnis umbenennen oder Dateien beziehungsweise Verzeichnisse innerhalb des Dateisystems bewegen.

Pfad: C:RENAME

Beschreibung:

RENAME ändert den Namen der nach FROM angegebenen Datei oder Verzeichnis in den nach TO eingegebenen. Dateien können nur innerhalb eines Geräts (einer Diskette oder Festplattenpartition) des Dateisystems umbenannt oder verschoben werden. Benennen Sie mit RENAME ein Verzeichnis um, wird dessen Inhalt nicht verändert.

Mit RENAME kann aber auch die Position eines Verzeichnisses oder einer Datei innerhalb der Datei-Hierarchie verändert werden.

```
RENAME :bernd/brief TO :maria/brief
```

verlegt die Datei »Brief« aus dem Verzeichnis »Bernd« in das Verzeichnis »Maria«, ohne deren Namen oder Inhalt dabei zu ändern. Auch mit ganzen Verzeichnissen ist das mög-

lich. Andere Betriebssysteme nennen diesen Vorgang »move«, also bewegen. Das ist zwar auch mit COPY und DELETE möglich. Bei RENAME wird aber die Datei nicht tatsächlich kopiert, sondern ihr Name nur anders eingeordnet. Das ist ein erheblicher Vorteil bei geringem freien Speicherplatz auf der Diskette, da die Datei während des Verlegens in ein anderes Verzeichnis nie zweimal auf der Diskette vorhanden ist, im Gegensatz zum Kopieren und danach Löschen einer Datei oder eines Verzeichnisses. Ferner nimmt RENAME weniger Zeit in Anspruch.

RENAME wird nicht ausgeführt, wenn die TO-Datei bereits existiert. Damit wird ein versehentliches Löschen von Dateien vermieden.

Beispiel(e):

```
RENAME arbeit/test AS :test/test-woanders
```

schreibt die Datei »test« aus dem Verzeichnis »arbeit« als Datei »test-woanders« in das Verzeichnis »test«. Das Verzeichnis test muß im Wurzelverzeichnis des aktuellen Geräts vorhanden sein.

RESIDENT

Format: RESIDENT <residenter Name> <Datei> [REMOVE] [ADD] [REPLACE] [PURE] [SYSTEM]

Muster: RESIDENT "NAME, DATEI, REMOVE/S, ADD/S, REPLACE/S, PURE/S, SYSTEM/S"

Zweck: Befehl resident machen.

Pfad: C:RESIDENT

Beschreibung:

RESIDENT erlaubt DOS-Befehle im Speicher des Amiga zu installieren, damit diese Befehle nicht vor jeder Ausführung neu in den Speicher geladen werden müssen. Dadurch wird bei der Bearbeitung mehrerer Aufgaben gleichzeitig (Multitasking) nicht nur die Ausführungszeit der Befehle kürzer, sondern es wird auch weniger Speicher verbraucht.

RESIDENT ist nur verfügbar, wenn die SHELL anstelle des CLI verwendet wird. Der Vorteil gegenüber der Speicherung eines Befehls in der RAM-Disk ist, daß das Programm beim Laden von der RAM-Disk noch einmal in ausführbarer Form in den Speicher des Rechners geladen wird, also mehr Speicherplatz beansprucht, da die RAM-Disk eine Diskettenstation im Speicher des Amiga simuliert.

Ein Befehl, der resident geladen werden kann, muß durch zwei oder mehr voneinander unabhängige Prozesse gleichzeitig benutzt und beliebig oft nacheinander ausgeführt werden können. DOS-Befehle, die diese notwendigen Eigenschaften aufweisen, besitzen ein gesetztes Schutzflag »p« (pure).

Suchen Sie mal mit LIST im Verzeichnis C: nach Befehlen, die das Schutzflag »p« gesetzt haben. Ist das p-Schutzflag nicht gesetzt, ist es wahrscheinlich, daß der Befehl nicht resident geladen werden kann. Wird es dennoch versucht, so ist das Risiko groß, daß der

Befehl abstürzt, wenn er ein zweites Mal gestartet wird, oder zweimal gleichzeitig aufgerufen wird.

RESIDENT ohne Optionen oder mit der Option REPLACE listet die Programme auf, die sich in der residenten Befehlsliste befinden. Die Option REPLACE ist standardmäßig eingeschaltet, und muß nicht extra angegeben werden.

Um einen Befehl neu in die Befehlsliste residenter Befehle einzufügen, gibt man seinen Namen und den vollständigen Zugriffspfad im Dateisystem an, um den Befehl zu erreichen, obwohl nur der Dateiname in die Liste aufgenommen wird.

Wurde der Name eines Befehls angegeben, der in der Systemliste für resident geladene Befehle eingetragen ist, wird versucht, diesen Befehl zu ersetzen. Wird der residente Befehl nicht von einem Prozeß verwendet, so kann er ersetzt werden, sonst wird der Versuch gestoppt.

Mit der Option SYSTEM wird der Befehl dem System-Teil der residenten Befehlsliste angefügt. Mit SYSTEM in die residente Befehlsliste aufgenommene Befehle, können bis zum nächsten Reboot nicht mehr aus dem System entfernt werden. Um SYSTEM-Dateien bei der Auflistung durch RESIDENT anzeigen zu lassen, muß zusätzlich die Option SYSTEM angegeben werden.

Mit PURE können Befehle, bei denen das p-Schutzflag nicht gesetzt ist, dennoch resident gemacht werden. Doch Vorsicht, machen Sie solche Befehle nur dann resident, wenn Sie gezielt testen wollen, ob zum Beispiel Eigenentwicklungen reentrant und multientrant sind, und sich somit zum resident-machen eignen.

Siehe auch: PROTECT

RUN

Format: RUN <Befehlsliste> [+ <Kommentar>]*

Muster: RUN "Befehl"

Zweck: RUN startet Prozesse (Programme), die im Hintergrund abgearbeitet werden.

Pfad: C:RUN

Beschreibung:

RUN erzeugt einen SHELL-Prozeß, der nicht im interaktiven Dialog abläuft. Der restliche Teil der Befehlszeile gilt als Eingabe für diesen Prozeß. Der SHELL-Prozeß führt die Befehle nacheinander im Hintergrund aus und beendet sich dann selbst. RUN zeigt die laufende Nummer des erzeugten Prozesses an.

Der neue SHELL-Prozeß hat das gleiche aktuelle Verzeichnis und denselben Befehls-Stapelspeicher wie die SHELL, von der RUN gestartet wurde. Mehrere Befehle, die im Hintergrund abgearbeitet werden sollen, können Sie durch ein Plus-Zeichen »+« und anschließende Betätigung der [Return]-Taste verbinden. RUN versteht die nächste Zeile, nach einem » + [Return]« als Fortsetzung der gleichen Zeile. Auf diese Weise ist es möglich, eine einzelne Befehlszeile aus mehreren Zeilen mit je einem Plus-Zeichen am Zeilenende zusammenzustellen.

Das SHELL-Fenster, von dem aus der Hintergrundprozeß gestartet wurde, wird normalerweise als Standardausgabe verwendet, die nicht geschlossen werden kann, solange ein ihr zugeordneter Prozeß aktiv ist. Wenn die Standardausgabe des RUN-Befehls nach NIL: umgeleitet wird, so kann das SHELL-Fenster nach dem Starten des Hintergrundprozesses mit ENDCLI geschlossen werden.

```
RUN >NIL ...
```

Wird der Hintergrundprozeß mit dem jeweils aktiven Fenster durch »*« verbunden, so kann dieses Fenster nicht geschlossen werden.

RUN überprüft zuerst, ob ein Befehl, den es starten soll, resident in den Speicher geladen worden ist, bevor es den Befehl auf dem mit PATH gesetzten Suchpfad sucht. Mit RUN kann auch ein neues SHELL-Fenster aktiviert werden, das zur Initialisierung die Script-Datei S:SHELL-Startup ausführt.

Beispiel:

```
RUN EXECUTE Befehle
```

führt die Script-Datei »Befehle« in einem Hintergrund-Prozeß aus.

```
RUN >NIL: Testprog
```

startet den Hintergrundprozeß so, daß das SHELL-Fenster, von dem aus der Hintergrundprozeß gestartet wurde, geschlossen werden kann.

SEARCH

Format: SEARCH [FROM] <Name>|<Muster> [SEARCH] <Zeichenkette> [ALL] [NONUM] [QUIET] [QUICK] [FILE]

Muster: SEARCH "FROM, SEARCH/A, ALL/S, NONUM/S, QUIET/S, QUICK/S, FILE/S"

Zweck: Suchen einer Zeichenkette innerhalb der angegebenen Textdateien.

Pfad: C:SEARCH

Beschreibung:

SEARCH sucht nach einer Zeichenkette in allen Dateien eines Verzeichnisses. SEARCH durchsucht auch alle Unterverzeichnisse, wenn Sie den Zusatz ALL anhängen. SEARCH gibt alle Zeilen aus, die die angegebene Zeichenkette beinhalten. Zudem wird der Name der Datei angezeigt, in der gerade gesucht wird. Eine Gruppe von Verzeichnissen kann durch Joker ausgewählt werden. Verwenden Sie einen Joker, durchsucht SEARCH Verzeichnisse, die der bezeichneten Schablone entsprechen. Näheres über Joker bei LIST.

Mit dem Schlüsselwort SEARCH, das angegeben werden muß, wird angezeigt, daß der folgende Parameter die zu suchende Zeichenkette ist. Amiga-DOS sucht die Zeichenkette entweder in Groß- oder Kleinschrift. Wenn die angegebene Zeichenkette Leerzeichen enthält, muß der Text in Anführungsstriche gesetzt werden. Falls notwendig, kann der Befehl

mit der Tastenkombination [Ctrl]+[C] abgebrochen werden. Um die Suche in der gerade durchsuchten Datei abzubrechen und in der nächsten Datei fortzusetzen dient [Ctrl]+[D].

Wird die geforderte Zeichenkette nicht gefunden, wird der Rückgabewert von SEARCH auf WARN gesetzt.

SEARCH ... NONUM	gibt nicht die Nummern der Zeilen aus, in denen die Zeichenfolge gefunden wurde.
SEARCH ... QUIET	zeigt keine Ausgaben an.
SEARCH ... QUICK	verwendet ein kompakteres Ausgabeformat.
SEARCH ... FILE	sucht eine Datei mit dem angegebenen Namen, statt eine Zeichenfolge in dieser Datei zu suchen.

Beispiel(e):

```
SEARCH arbeit SEARCH test
```

Durchsucht im Verzeichnis »arbeit« alle Dateien nach dem String »test«.

```
SEARCH DF1: SEARCH test ALL
```

Durchsucht alle Dateien und alle Dateien in Unterverzeichnissen auf DF1: nach der Zeichenkette »test«.

SETCLOCK

Format: SETCLOCK [LOAD | SAVE | RESET]

Muster: SETCLOCK "LOAD/S,SAVE/S,RESET/S"

Zweck: Übergabe der Systemzeit an die Hardware-Uhr oder Übernahme der Systemzeit von der Hardware in die Systemuhr.

Pfad: C:SETCLOCK

Beschreibung:

Glückliche Besitzer eines Amiga 2000 oder eines Amiga 500 mit Speichererweiterung verfügen in ihrem Rechner über eine batteriegepufferte Echtzeituhr zusätzlich zur systeminternen Uhr des Amiga. Diese batteriegepufferte Uhr kann mit dem Programm »SYS:Prefs/Preferences« zusammen mit der Systemuhrzeit gestellt werden.

Unabhängig davon bietet SETCLOCK die Möglichkeit, mit der Option SAVE die Uhrzeit der Echtzeituhr allein nach der Systemuhrzeit zu stellen, und mit der Option LOAD die Systemuhr nach der Echtzeituhr zu stellen.

```
SETCLOCK SAVE
```

Die Systemzeit wird in die Hardware-Uhr übertragen, und selbst ein Ausschalten des Rechners läßt die Echtzeituhr weiterlaufen.

Bei einem Reboot des Rechners kann nun die Systemuhr, die bei einem Reboot auch zurückgesetzt wird, auf die richtige Zeit gestellt werden. Das geschieht mit dem Kommando:

```
SETCLOCK LOAD
```

Es empfiehlt sich, dieses Kommando in die Datei S:Startup-Sequenz zu übernehmen, so daß die Uhrzeit aus der batteriegepufferten Echtzeituhr bei jedem Boot-Vorgang auch in die Systemuhr übernommen wird.

Mit der Option RESET kann die Uhr vollständig zurückgesetzt werden, wenn im Falle eines Falles durch den Fehler eines Programmes die Uhr ausgeschaltet, oder mysteriöse Effekte hervorgerufen werden.

Bei ziemlich alten Versionen der Workbench-Diskette, die Sie zusammen mit Ihrem Computer bekommen haben, kann der entsprechende Befehl zum Setzen der Zeit auch SETTIME heißen.

SETENV

Format: SETENV <Variablenname> <Zeichenfolge der Variablen>

Muster: SETENV "NAME/A, ZEICHENFOLGE"

Zweck: Setzen einer Umgebungsvariablen.

Pfad: C:SETENV

Beschreibung:

SETENV dient zum Festlegen des Wertes einer Umgebungsvariablen. Umgebungsvariablen werden im logischen Gerät ENV: gespeichert, das in der Datei »S:Startup-Sequence« initialisiert werden sollte.

Der Inhalt einer Umgebungsvariablen wird mit SETENV <Variablenname> ohne Angabe einer Zeichenfolge gelöscht. Die Variable bleibt in ENV: vorhanden, ist jedoch leer.

Beispiele:

```
SETENV EDITOR Extras1.3D:Tools/MEMACS
```

Weist der Umgebungsvariablen »EDITOR« den Wert »Extras1.3D:Tools/MEMACS« zu. MORE benutzt die Variable EDITOR um festzustellen, welches Programm als Editor für MORE eingesetzt wird. In diesem Fall wird der Editor MEMACS benutzt.

Siehe auch: GETENV

SETPATCH

Format: SETPATCH [R]

Muster: SETPATCH "R/S"

Zweck: Korrektur von Fehlern im ROM der Kickstart-Versionen 1.2 und 1.3.

Pfad: C:SETPATCH

Beschreibung:

Die Kickstart-ROMs der Version 1.2 und 1.3 weisen einige bekannte Fehler auf, die SETPATCH korrigiert. SETPATCH muß als erster Befehl in der Datei S:Startup-Sequence ausgeführt werden, so wie das bei der Original-Workbench1.3-Diskette der Fall ist.

SETPATCH beseitigt die Fehler von:

- `Display-Alert()`
- `Delete-Layers()`
- `Alloc-Entry()`
- `der Bearbeitung der mathematischen Exeption-Vektoren des MC68000-Prozessors`

Die Option R wird verwendet, damit bei Systemen mit 1 Mbyte CHIP-RAM die resetfeste RAM-Disk bei einem Reboot nicht gelöscht wird. Die neueste Revision des Motherboards (6.x) enthält eine neue Version des Fat-Agnus-Chip, die ein Megabyte Hauptspeicher als CHIP-RAM zur Verfügung stellt.

SETMAP

Format: SETMAP <Tastaturbelegung>

Zweck: Ändert die landesabhängige Tastaturbelegung.

Pfad: SYSTEM:SETMAP

Beschreibung:

Es existieren mehrere alternative Tastaturbelegungen, die an den Gebrauch in den verschiedenen europäischen Ländern angepaßt sind. Das Verzeichnis »DEVS:KEYMAPS« enthält die verfügbaren Tastaturbelegungen. Um auf die deutsche Tastaturbelegung umzuschalten, geben Sie bitte ein:

```
SETMAP D
```

Sie können dieses Kommando selbstverständlich in Ihre Startup-Sequenz übernehmen. Auf der Diskette EXTRAS1.3D sind im Ordner »DEVS/KEYMAPS« verschiedene Tastaturbelegungen zu finden.

SORT

Format: SORT [FROM]<name> [[TO]<name>] [COLSTART<n>]

Muster: SORT "FROM/A, TO/A, COLSTART/K"

Zweck: Textdateien zeilenweise sortieren.

Pfad: C:SORT

Beschreibung:

Mit SORT lassen sich Dateien sortieren. Dieser Befehl ist allerdings für lange Dateien nicht schnell genug und er kann keine Dateien sortieren, die nicht in den Speicher passen. Sie bezeichnen die zu sortierende Datei als FROM-Datei, das Ergebnis geht in die Datei, die Sie hinter TO angeben. SORT geht davon aus, daß die FROM-Datei eine normale

Textdatei ist, in der jede Zeile mit einem [Return] beendet wurde. Die Datei wird zeilenweise alphabetisch sortiert, ohne Rücksicht auf Groß- oder Kleinschreibung. Um dies in beschränktem Maß zu umgehen, geben Sie das Schlüsselwort COLSTART ein, um diejenige Spalte festzulegen, ab der der Vergleich stattfinden soll. SORT vergleicht dann die Buchstaben der Zeile ab dem bezeichneten Startpunkt bis zum Zeilenende. Lassen sich die Zeilen so nicht eindeutig ordnen, werden die Zeichen vor der mit COLSTART bezeichneten Spalte mit einbezogen.

Achtung! Der normalerweise eingerichtete Stapelspeicher ist nur 4000 Byte groß. Das reicht höchstens zum Sortieren von Dateien mit 50 Zeilen Länge. Wollen Sie größere Dateien sortieren, müssen Sie den Stack mit dem Befehl STACK vergrößern. Wieviel Stack Sie brauchen, hängt von der Größe der Datei ab und kann nur durch Schätzen ermittelt werden, im allgemeinen sollte der Stapelspeicher (Stack) ungefähr doppelt so groß sein, wie die zu sortierende Datei. Haben Sie den Stack aber zu klein gemacht, verweigert der Amiga jede weitere Zusammenarbeit (Absturz) bis zum Neustart mit [Ctrl]+[Linke Amiga] +[Rechte Amiga].

Beispiel(e):

```
SORT test TO sorttest
```

sortiert die Datei »test« zeilenweise und schreibt das Ergebnis in die Datei »sorttest«.

```
SORT index TO sortierter.index COLSTART 4
```

sortiert die Datei »index« ab der vierten Spalte und schreibt das Ergebnis in die Datei »sortierter.index«.

Siehe auch: STACK

STACK

Format: STACK [<n>]

Muster: STACK "GRÖSSE"

Zweck: Verändern der Stack-Größe.

Pfad: C:STACK

Beschreibung:

STACK zeigt die Größe des Stapelspeichers an, oder läßt die Änderung dessen Größe zu. Wenn ein Programm abläuft, benötigt es einen gewissen Speicherplatz zum Zwischenspeichern von Werten und um programminterne Sprünge durchführen zu können. In den meisten Fällen reicht der eingestellte Speicherplatz aus, trotzdem kann er mit dem Befehl STACK verändert werden. Dies erreicht man durch den Befehl STACK, gefolgt von der neuen Stapelspeichergröße in Byte. Der Befehl STACK ohne Argument zeigt die aktuelle Größe des Stapel-Speichers an.

SORT ist ein Befehl, für den man die Größe des Stapelspeichers verändern muß. Rekursive Befehle wie zum Beispiel DIR benötigen allerdings ebenfalls einen erweiterten Stack, wenn mehr als sechs ineinander verschachtelte Verzeichnisse auftreten. Oft benötigen auch Compiler einer Programmiersprache einen vergrößerten Stapelspeicher.

Achtung! Zu wenig Stack läßt den Amiga abstürzen, so daß nur noch ein Reboot weiterhilft. Machen Sie den Stack lieber zu groß als zu klein!

Beispiel(e):

```
STACK 8000
```

setzt die Stack-Größe auf 8000 Byte.

```
STACK
```

zeigt die Größe des aktuellen Stacks an.

STATUS

Format: STATUS [<Prozeß>] [FULL] [TCB] [CLI|ALL] [COMMAND]

Muster: STATUS "PROZESS, FULL/S, TCB/S, CLI=ALL/S COMMAND/K"

Zweck: Informationen über SHELL-Prozesse ausgeben.

Pfad: C:STATUS

Beschreibung:

Der Befehl gibt Informationen über alle momentan gestarteten SHELL-Prozesse aus. STATUS alleine listet die Prozeßnummern aller arbeitenden Programme auf.

<Prozeß> bezeichnet die Nummer eines bestimmten SHELL-Prozesses, über den Informationen ausgegeben werden sollen. Das Schlüsselwort FULL führt Funktionen der unten erklärten Schlüsselwörter TCB und ALL (oder CLI) kombiniert aus. TCB zeigt Informationen über Priorität, Stack-Größe und globale Vektorgröße aller von der SHELL gestarteten Prozesse. Weitere Details zu Stack und globaler Vektorgröße finden Sie im AmigaDOS-Programmiererhandbuch.

Das Schlüsselwort CLI (ALL) listet SHELL-Prozesse und zeigt die Namen der gerade laufenden Befehle, falls vorhanden.

Die Option COMMAND=<Name> weist STATUS an, die Liste der Prozesse nach einem bestimmten Programm namens »Name« zu durchsuchen. Wird das Programm gefunden, wird seine Prozeßnummer ausgegeben und der Rückgabewert auf Null gesetzt. Wird das Programm nicht gefunden, so wird WARN zurückgegeben.

Beispiel(e):

```
STATUS
```

gibt Informationen über alle Prozesse.

```
STATUS 4 FULL
```

gibt alle Informationen über den SHELL-Prozeß 4.

Soll BREAK in einer Script-Datei an den Prozeß gesendet werden, der den Befehl ClockPtr ausführt, lautet der Ausdruck:

```
STATUS >t:help_brk COMMAND=ClockPtr
BREAK  <t:help_brk >NIL: ?
```

TYPE

Format: TYPE [FROM]<name> [[TO]<name>] [OPT N=NUMBERS|H=HEX]

Muster: TYPE "FROM/A, TO/S, OPT/K, HEX/S, NUMBER/S"

Zweck: Ausgabe einer Datei.

Pfad: C:TYPE

Beschreibung:

Der Befehl TYPE gibt eine Textdatei oder ein Programm aus. Dabei werden Tabulatoren, die in der Datei vorhanden sind, in der Ausgabe als acht Zeichen breite Einrückungen dargestellt.

Die Option TO bezeichnet die Zieldatei. TYPE überprüft, ob die angegebene Zieldatei bereits auf dem Speichermedium vorhanden ist, und gibt, wenn die Datei schon besteht, eine Fehlermeldung aus. Wurde das Schlüsselwort TO explizit angegeben, wird die vorhandene Datei überschrieben. Lassen Sie die Ausgabedatei fort, geht die Ausgabe auf das aktuelle Ausgabegerät, das heißt in den meisten Fällen in das SHELL-Fenster, in dem TYPE aufgerufen wurde.

Die Ausgabe kann durch Drücken von [Ctrl]+[C] abgebrochen und durch Drücken der Leerzeichen- oder irgendeiner anderen Taste angehalten werden. Betätigen Sie die [Return]-Taste, [Backspace] oder [Ctrl]+[X], wird die Ausgabe danach wieder fortgesetzt.

Mit OPT N oder NUMBERS sind zusätzliche Modifikationen der Ausgabe möglich. NUMBERS sorgt dafür, daß Zeilennummern mit ausgegeben werden, die HEX bewirkt die Ausgabe der Datei als Hexadezimalzahlen und Klartext.

Beispiel(e):

```
TYPE arbeit/text
```

schreibt den Inhalt der Datei »text« auf den Bildschirm.

```
TYPE arbeit/text NUMBERS
```

schreibt den Inhalt von »text« mit Zeilennummern auf den Bildschirm.

```
TYPE Programm TO Hexdump HEX
```

schreibt die Datei »Programm« in hexadezimaler Form in die Datei »Hexdump«.

VERSION

Format: VERSION <Bibliotheksname|Gerätename> [<Versionsnr.>] [<Revisionsnr.>] [<Einheitennr.>]

Schablone: VERSION "NAME,VERSION, REVISION, EINHEIT"

Zweck: Auffinden der Versions- und Revisionsnummer einer Workbench-Diskette.

Pfad: C:VERSION

Beschreibung:

VERSION wird verwendet, um die Versions- und Revisionsnummer einer Bibliothek, eines Geräts oder der Workbench-Diskette anzuzeigen. VERSION kann auch auf spezielle Versions-/Revisionsnummern untersuchen und WARN zurückgeben, falls die gefundenen Versions-/Revisionsnummern höher sind als die angegebenen.

VERSION ohne das Argument <Bibliotheksname|Gerätename> gibt die Versionsnummer des Kickstart und der Workbench aus. Wurde <Bibliotheksname|Gerätename> eingegeben, versucht der Befehl VERSION, die Bibliothek oder das Gerät zu öffnen und die Versions-Informationen zu lesen.

Wurden <Versionsnr.> und <Revisionsnr.> angegeben, setzt VERSION den Rückgabewert mit 0, falls die gefundenen Versions- und Revisionsnummern höher als die angegebenen Werte oder gleich sind. Andernfalls wird der Fehlercode mit 5 (=WARN) gesetzt. Wurde keine Revisionsnummer eingegeben, erfolgt kein Vergleich der Revisionsnummern.

Der Befehl VERSION kann nicht zusammen mit einem Pfadnamen verwendet werden, um die Version eines Geräts zu prüfen.

Die Option <Einheitennr.> erlaubt eine Einheitennummer, die ungleich 0 ist, anzugeben. Dies ist beim Zugriff auf Geräte, die aus mehreren Einheiten bestehen (z.B. SCSI-Schnittstelle), gelegentlich erforderlich.

WAIT

Format: WAIT <n> [SEC=SECS][MIN=MINS] [UNTIL <zeit>]

Muster: WAIT "SEC=SECS/S, MIN=MINS/S, UNTIL/K"

Zweck: WAIT wartet eine bestimmte Zeit.

Pfad: C:WAIT

Beschreibung:

Der Befehl WAIT verzögert eine Befehlssequenz oder einen mit RUN gestarteten Prozeß für eine gewisse Zeitspanne oder bis zu einem bezeichneten Zeitpunkt. Falls Sie nichts anderes eingegeben haben, wartet der Amiga eine Sekunde lang. Der eingegebene Parameter ist eine Zahl (auch mit führender Null), die der Wartezeit in Sekunden (oder Minuten, falls das Schlüsselwort MINS eingegeben wird) entspricht. Verwenden Sie das Schlüsselwort UNTIL, wartet das System bis zu einem gewissen Zeitpunkt, der in der Form SS:MM (Stunde:Minute) eingegeben wird; genau bis die angegebene Minute halb verstrichen ist.

Beispiel(e):

```
WAIT
```

hält die Ausführung für eine Sekunde an.

```
WAIT 10 MINS
```

hält die Ausführung 10 Minuten an.

```
WAIT UNTIL 21:15
```

wartet mit der weiteren Ausführung der Befehle, bis die Systemuhr 21.15 Uhr zeigt.

WHICH

Format: WHICH <Dateiname> [NORES] [RES]

Muster: WHICH "DATEI/A, NORES/S,RES/S"

Zweck: Absuchen des Befehlspfads nach einem bestimmten Befehl.

Pfad: C:WHICH

Beschreibung:

WHICH hilft beim Suchen eines bestimmten DOS-Befehls, indem der Zugriffspfad zu dem Befehl im Falle des Vorhandenseins ausgegeben wird. WHICH sucht nach dem angegebenen Befehl normalerweise in der residenten Befehlsliste, im aktuellen Verzeichnis, dem/den Befehlspfad(en) und dem Verzeichnis C:. Wird er nicht gefunden, wird der Rückgabewert WARN zurückgegeben.

Die Option NORES verhindert die Suche in der residenten Befehlsliste nach dem Befehl.

Mit RES wird WHICH veranlaßt, nur in der residenten Befehlsliste zu suchen.

Beispiel(e):

```
WHICH DIR

DH0:C/DIR
```

WHY

Format: WHY

Muster: WHY

Zweck: Ausgabe weiterer Informationen über einen aufgetretenen Fehler.

Beschreibung:

Tritt bei der Bearbeitung eines DOS-Befehls ein Fehler auf, bricht der Amiga die Ausführung des Befehls ab und gibt eine Fehlermeldung aus. Diese enthält normalerweise den Namen der Datei (falls eine Datei den Fehler bedingte), ohne weitere Informationen an. Konnte zum Beispiel der Befehl:

```
COPY fred TO *
```

nicht ausgeführt werden, erscheint die Meldung:

```
Can't open fred
```

Warum »fred« nicht geöffnet werden konnte, erfährt der Anwender nicht. So könnte »fred« zum Beispiel ein Verzeichnis sein, oder der Speicher des Amiga könnte voll sein, oder eine Diskette könnten keinen freien Speicherplatz mehr haben, wenn in eine Datei geschrieben werden sollte.

Wird WHY (warum) eingegeben, bevor ein weiterer DOS-Befehl gestartet wird, hilft WHY herauszufinden, was nicht stimmte.

Beispiel(e):

```
TYPE DF0:
Can't open DF0:
WHY
Last command failed because objekt is not of required type
```

»Der letzte Befehl konnte nicht ausgeführt werden, weil das Objekt nicht vom benötigten Typ ist«. Dieser Befehl konnte also nicht ausgeführt werden, weil ein Gerät nicht auf dem Bildschirm ausgegeben werden kann. WHY gibt also auch nur Hinweise, keine fertigen Lösungen aus.

4 Batchprogrammierung

4.0 Einführung in Script-(Batch)-Dateien

Selbst wenn ein Anwender die Funktion aller DOS-Befehle gelernt hat und regelmäßig auf der SHELL arbeitet, ist es selten, daß er versucht, sich seine Arbeit mit kleinen Dienstprogrammen zu erleichtern, die er sich aus den DOS-Befehlen selbst zusammenbasteln kann.

In einer normalen Arbeitssitzung auf der SHELL arbeitet der Benutzer fast nur interaktiv; tippt seine Befehle ein und wartet auf das Ergebnis. Manchmal startet er auch mit RUN Hintergrundprozesse. Man gewöhnt sich an umständliche routinemäßige Arbeitsprozesse, oder ärgert sich fortwährend über Leistungsmerkmale, die bei einem Befehl leider nicht vorhanden sind. Hier kann uns der Amiga viel Arbeit abnehmen.

Es mag sein, daß sich mancher Anwender einen wesentlich komfortableren LIST-Befehl gewünscht hat, als den im C-Ordner befindlichen.

Mit einer kleinen Script-Datei geht dieser Wunsch in Erfüllung:

```
.K file
STACK 32768 ;notwendig für SORT (overkill)
LIST >T: <$$>list_tmp <file> NOHEAD
SORT FROM T: <$$>list_tmp TO T: $$>sort_tmp
MORE T: <$$>sort_tmp
DELETE T: <$$>list_tmp T: <$$>sort_tmp
STACK 4000 ;alter Wert des Stack
```

Nachdem diese Datei mit einem Texteditor wie Micro-Emacs (MEMACS) von der Extras-Diskette erstellt und unter dem Namen LISTER in das Verzeichnis S: gespeichert worden ist, kann sie mit

```
EXECUTE S:LISTER <Verzeichnis>
```

ausgeführt werden. Wird das Script-Flag mit

```
PROTECT S:LISTER +S
```

gesetzt, kann die Script-Datei wie ein DOS-Befehl allein mit ihrem Namen aufgerufen werden. Was tut diese Script-Datei? Zuerst wird »file« als Platzhalter für einen Verzeichnisnamen vereinbart. Dann wird sicherheitshalber wegen des verwendeten SORT-Befehls der Stapelspeicher auf 32 Kbyte heraufgesetzt, um auch bei langen Inhaltsverzeichnissen Abstürze von SORT zu vermeiden (siehe auch SORT). Danach wird eine Liste aller Dateien und Verzeichnisse im aktuellen oder angegebenen Verzeichnis erstellt und in eine

temporäre Datei im T-Verzeichnis geschrieben. Mit SORT wird anschließend der Inhalt sortiert in eine zweite temporäre Datei geschrieben und mit MORE angezeigt, so daß ein Vor- und Zurückblättern im Inhaltsverzeichnis möglich wird. Zuletzt werden die temporären Dateien wieder gelöscht und der Stapelspeicher auf seine Normalgröße zurückgesetzt.

Hoffentlich wurde mit dem Beispiel die Neugier des ein oder anderen darauf geweckt, was in Script-Dateien denn noch alles möglich sei. Zuerst werden die Befehle erklärt, mit denen man den Ablauf von Script-Dateien ähnlich wie in einer Programmiersprache steuern kann, um zum Teil sehr komplexe Probleme zu lösen.

4.1 Starten und Parameterübergabe

EXECUTE

Format: EXECUTE <Script-Datei> [<argumente>*]

Muster: EXECUTE "SCRIPT-DATEI","ARGUMENTE"

Zweck: Ausführen einer Befehlsdatei mit Argumentübergabe.

Beschreibung:

EXECUTE dient zum Einsparen von Tipparbeit. Eine Script-Datei enthält Anweisungen, die vom Kommadozeilen-Interpreter ausgeführt werden können (also alle Befehle, die wir hier besprechen). Amiga-DOS führt diese Befehle einen nach dem anderen aus, genau in der Reihenfolge, wie sie in der Befehls-Datei stehen. Der Befehl EXECUTE wird nicht unbedingt benötigt; wenn bei den Schutzflags der Script-Datei mit PROTECT +S das Script-Flag gesetzt wurde, braucht EXECUTE vor dem Namen der Datei nicht mehr eingetippt werden (siehe auch: PROTECT).

EXECUTE erzeugt ein Verzeichnis mit dem Namen »SYS:T«, sofern ein logisches Gerät »T:« noch nicht existiert.

EXECUTE kann auch verwendet werden, um Parameter an die einzelnen Befehle der Script-Datei zu übergeben, wobei Sie verschiedene Werte als Parameter angeben können. Bevor die Befehlsdatei ausgeführt wird, ersetzt Amiga-DOS die Platzhalter für die Parameter in der Befehlsdatei mit denen, die Sie nach dem Befehl EXECUTE angegeben haben.

Auf die aktuelle Befehlsprozessor-Nummer kann durch den Platzhalter <$$> in der Script-Datei zugegriffen werden. Dies ist hilfreich beim Erzeugen eindeutiger Namen für temporäre Dateien, logische Zuweisungen und PIPE-Namen (siehe Beispiel in der Einleitung zu diesem Kapitel).

Für Platzhalter können vorbelegte Werte angegeben werden, die Amiga-DOS verwendet, wenn beim Aufruf der Script-Datei keine Werte für die Platzhalter angegeben wurden. Haben Sie hinter EXECUTE keine Parameter angegeben, obwohl die Script-Datei solche erwartet, und sind keine Voreinstellungen für diese Werte vorhanden, dann ist der Wert des jeweiligen Parameters nicht belegt, und es wird nichts dafür eingesetzt.

```
.KEY                Kennzeichnung, daß nun die Definition der ver-
                    schiedenen Argumente folgt. Abgekürzt mit .K
.DOT ch             Ändert das Punkt-Zeichen ».« in ch.
.BRA ch             Ändert das Kleiner-Zeichen »<« in ch.
.KET ch             Ändert das Größer-Zeichen »>« in ch.
.DOLLAR ch          Ändert das Voreingestellt-Zeichen »$« in ch,
                    abgekürzt als .DOL
.DEF Schlüsselwort  Gibt globalen Ersatzwert an Parameter
.<Leerzeichen>      Kommentarzeile
.<Neue Zeile>       Leerzeile
```

Vor der Ausführung durchsucht Amiga-DOS den Inhalt der Script-Datei nach Platzhaltern, die von den Kleiner- und Größer-Zeichen »<« und »>« eingeschlossen werden. Solche Begriffe können ein Schlüsselwort (also den Namen eines Übergabeparameters) oder das Schlüsselwort und ein Ersatzwert (= Default-Wert) beinhalten, für den Fall, daß der jeweilige Parameter nicht angegeben wurde. Um das Schlüsselwort und sein Ersatzwort, falls vorhanden, zu trennen, wird das Dollar-Zeichen »$« verwendet. Auf diese Weise ersetzt Amiga-DOS den Ausdruck »<TIER>« durch den Wert, den Sie hinter EXECUTE für das Schlüsselwort »TIER« angegeben haben. Das Gleiche passiert mit dem Ausdruck »<TIER$KATZE>«. Haben Sie für TIER jedoch keinen Wert übergeben, so wird als Default-Wert einfach KATZE eingesetzt.

Eine EXECUTE-Datei kann Punkt-Befehle nur verwenden, wenn in der ersten Zeile ein Punkt-Befehl steht! Der Kommandozeilen-Interpreter untersucht die erste Zeile, und wenn diese mit einem Punkt-Befehl (beispielsweise einem Kommentar) beginnt, durchsucht der Kommandozeilen-Interpreter die Datei auf Platzhalter und erzeugt eine temporäre Datei, die im T-Verzeichnis abgelegt wird. Wenn dagegen die Datei nicht mit einem Punkt-Befehl beginnt, wird angenommen, daß sich keine Punkt-Befehle in ihr befinden und damit auch, daß keine Platzhalter ersetzt werden sollen.

Für den Fall, daß sich kein Punkt-Befehl in der ersten Zeile befindet, beginnt der Kommandozeilen-Interpreter die Datei auszuführen, ohne sie zunächst in das Verzeichnis T: zu kopieren, was sonst passieren würde.

Zu beachten ist, daß Kommentare auch in Script-Dateien möglich sind, indem das Kommentar-Zeichen, das auf der SHELL verwendet wird, das Semikolon (;), vor den Kommentartext gesetzt wird.

Amiga-DOS stellt eine Reihe von Befehlen zur Verfügung, die nur in Script-Dateien sinnvoll sind, um den Ablauf der Script-Dateien zu steuern, die alle in diesem Kapitel vorgestellt und erläutert werden.

Es können auch weitere Script-Dateien innerhalb einer Script-Datei aufrufen werden. Das heißt, Script-Dateien können EXECUTE-Befehle beinhalten. Um die Ausführung einer Befehlsdatei zu unterbrechen, muß [Ctrl]+[D] gedrückt werden. Mit [Ctrl]+[C] wird die Ausführung ineinander verschachtelter EXECUTE-Dateien vorzeitig beendet. [Ctrl]+[D] beendet nur die jeweils aktuelle Datei. Die Abbruchsignale können natürlich auch mit BREAK an die Script-Prozesse gesendet werden.

Beispiel(e):

Angenommen, das File »list« sieht so aus:

```
.K filename/A
RUN COPY <filename> TO PRT:+
ECHO "<filename> ausgedruckt!"
```

Dann ergibt der Befehl

```
EXECUTE list testprogramm
```

das gleiche Ergebnis wie die direkte Eingabe von:

```
RUN COPY testprogramm TO PRT:+
ECHO "testprogramm ausgedruckt"
```

Noch ein etwas umfangreicheres Beispiel. Das Befehlsfile heißt diesmal »ANZEIGER«:

```
.KEY name/A
IF EXISTS <name>
TYPE <name> NUMBERS
ELSE
ECHO "<name> steht nicht in diesem Verzeichnis"
ENDIF
```

Der Befehl

```
EXECUTE anzeiger testfile
```

gibt das File »testfile« mit Zeilennummern aus, wenn es im aktuellen Verzeichnis steht, ansonsten erscheint die Meldung:

```
»testfile steht nicht im aktuellen Verzeichnis«
```

Parameterübergabe durch Schlüsselwörter und/oder durch die Position der Parameter in der Befehlszeile.

Der Ausdruck .KEY (oder abgekürzt auch .K) berücksichtigt sowohl die unterschiedlichen Namen der Schlüsselwörter als auch die Reihenfolge der Argumente. Er teilt dem Befehl EXECUTE mit, wieviele Argumente hinter EXECUTE erwartet werden müssen und wie sie zu interpretieren sind. Mit anderen Worten ist der .KEY-Befehl mit allem, was dahinter angegeben wird, eine Art Eingabe-Schablone für die Parameter, die nach dem Namen der Script-Datei angegeben werden können. Es ist nur eine .KEY-Zuweisung pro Befehlsdatei erlaubt. Wird von der Möglichkeit Gebrauch gemacht, so muß sie in der ersten Zeile der Befehlsdatei stehen.

Nehmen wir also an, unsere erste Script-Datei mit dem Namen »demo1« enthält als erste Zeile folgende .KEY-Zuweisung:

```
.KEY blitz, pfanne
```

Amiga-DOS weiß nun bei der Ausführung der Datei, daß zwei Argumente zu übernehmen sind, nämlich <blitz> und <pfanne>. Angenommen, Sie geben nun in Ihrem CLI-Fenster interaktiv folgende Befehlszeile ein (beachten Sie, daß die Script-Datei hier den Namen DEMO1 besitzt):

```
EXECUTE DEMO1 pfanne einname blitz anderername
```

dann wird der Name »einname« dem Schlüsselwort »pfanne« und der Name »anderername« dem Schlüsselwort »blitz« zugeordnet. Durch Einhalten der in dem .KEY-Befehl gegebenen Reihenfolge der Parameter kann jedoch die Angabe der Schlüsselworte wegfallen. Folgende Befehlszeile erfüllt den gleichen Zweck wie die oben gezeigte:

```
EXECUTE DEMO1 anderername einname
```

Die Reihenfolge hält sich an das in der .KEY-Befehlszeile angegebene Muster. Beide Formen der Parameterübergabe können auch gemischt angewendet werden. Der nächste .KEY-Befehl bestimmt zum Beispiel vier Argumente:

```
.KEY wort1, wort 2, wort3, wort4
```

Der folgende EXECUTE-Befehl führt dann durchaus zum richtigen Ergebnis (unsere Befehlsdatei heißt hier DEMO2):

```
EXECUTE DEMO2 wort3 ccc wort1 aaa bbb ddd
```

Amiga-DOS ordnet den Wert ccc vereinbarungsgemäß dem Parameter <wort3> zu. <wort1> erhält den Wert aaa. Der nächste unbelegte Parameter ist also <wort2> und dem wird der Wert bbb zugewiesen. Der letzte Wert ddd bleibt dann noch für den letzten unbelegten Parameter <wort4> übrig. Jeder einzelne Parameter kann noch zusätzlich konditioniert werden:

```
.KEY name1/A, name2/A, name3, name4/K
```

Das »/A« bedeutet, daß den Parametern <name1> und <name2> in jedem Fall Werte übergeben werden müssen. Werte für die Parameter <name3> und <name4> können weggelassen werden, sie sind nicht unbedingt erforderlich. Andererseits legt das »/K« von Parameter <wort4> fest, daß die Angabe des Schlüsselwortes (hier name4) zwingend vorgeschrieben ist.

```
EXECUTE DEMO3 eins zwei drei NAME4 vier
```

ist eine für die obige Befehlsdatei gültige Wertzuweisung. Ein Fehler bei der Parameter-Übergabe bricht die Ausführung der Script-Datei mit einer Fehlermeldung ab. Hier noch ein Beispiel für die Option »/k«:

Eine Script-Datei COMPILIERE soll einen Programmtext »Quelltext« in den Binärcode des Rechners übertragen und den Binärcode in eine Datei »Ausführbar« schreiben, wenn dies gewünscht wird. Der .KEY-Befehl könnte so aussehen:

```
.k Eingabe/A, Programmname/K
```

Geben Sie dann diese Befehlszeile ein:

```
EXECUTE COMPILIERE Quelltext PROGRAMMNAME Ausführbar
```

Nur so wird der Befehl korrekt ausgeführt, da die optionale Ausgabe des Quelltextes in die Datei »Ausführbar« mit Schlüsselwort und Parameter angefordert wurde.

Wertzuweisung an vorbelegte Platzhalter und Umdefinition der Klammerzeichen:

Mit dem Befehl .DEF wird einem Platzhalter, der im Aufruf der Script-Datei vom Benutzer keinen Wert zugewiesen bekommt, ein Ersatzwert zugewiesen. Dieser Ersatzwert muß zugewiesen werden, bevor der Parameter das erste Mal verwendet wird. Der Ersatzwert wird in Anführungszeichen ("") gesetzt oder durch das Dollarzeichen ($) getrennt direkt hinter dem zu ersetzenden Platzhalter in der Befehlsdatei eingegeben. Bei dieser Möglichkeit muß der Ersatzwert bei jeder Verwendung dieses Schlüsselwortes neu eingegeben werden.

```
.K wort1
ECHO "<wort1$ersatzwort1> Ersatz für wort1."
ECHO "<wort1> 2.Versuch"
.DEF wort1 "ersatzwort1"
ECHO "<wort1> ist der Ersatz für wort1."
ECHO "<wort1> 2.Versuch"
```

Wenn kein Ersatzparameter angegeben wird erscheint folgende Ausgabe:

```
ersatzwort1 ist der Ersatz für wort1
2.Versuch
```

wort1 hat zu diesem Zeitpunkt keinen Inhalt mehr!

```
ersatzwort1 ist der Ersatz für wort1
ersatzwort1 2.Versuch
```

wort1 wurde durch .DEF global auf den Wert »ersatzwort1« gesetzt.

Definition anderer Klammerzeichen:

Wann immer EXECUTE ein Schlüsselwort in spitzen Klammern vorfindet, sucht es den passenden Wert dazu. Ein nicht definierter Parameter ohne Voreinstellung bekommt einen Leerstring zugewiesen. Sollen die spitzen Klammern innerhalb einer Zeichenkette als Zeichen verwendet werden, können Ersatzzeichen für die Klammern definiert werden:

.BRA und .KET sind die Befehle zum neu Belegen der spitzen Klammern. Das folgende Befehlsfile »test« nutzt diese Möglichkeit:

```
.KEY wort1
ECHO "Diese Zeile druckt KEINE <spitzen> Klammern!"
.BRA {
.KET }
ECHO "Diese Zeile DRUCKT <spitze> Klammern!"
ECHO "Das Ersatzwort ist {wort1}."
```

Mit dem folgenden EXECUTE-Befehl:

```
EXECUTE TEST ersatzwort
```

ergibt das dann folgende Ausgabe am Bildschirm:

```
Diese Zeile druckt KEINE Klammern!
Diese Zeile DRUCKT <spitze> Klammern!
Das Ersatzwort ist ersatzwort.
```

Der erste ECHO-Befehl suchte nach einem Argument für den Parameter »eckige«, konnte keines finden und teilte dem Parameter einen Leerstring zu. Dann wurden die geschweiften Klammern als Ersatz für die eckigen Klammern definiert, die ihrerseits damit zu normal druckbaren Zeichen wurden, was der zweite ECHO-Befehl bewies. Der dritte ECHO-Befehl beweist die Funktion der neuen Parameter-Kennung.

Beispiel 3:

Programmstrukturen in EXECUTE-Files

Der Befehl IF (zu deutsch: falls) erlaubt die Ausführung eines weiteren Befehls in Abhängigkeit einer erfüllten oder nicht erfüllten Bedingung. Mit diesem Befehl können zum Beispiel zwei Strings auf Gleichheit oder die Existenz eines Strings überprüft werden. Der Befehl ELSE schreibt den weiteren Ablauf des Programmes vor, wenn die IF-Bedingungen nicht erfüllt sind. Der Befehl ENDIF beendet den Block der IF-Befehle. Sie kennen das sicher aus der Programmiersprache Basic und vielen anderen Sprachen. Im folgenden Beispiel wird zusätzlich der Befehl SKIP (Sprung) verwendet. Dieser Befehl erlaubt nur Sprünge mit LAB definierten Programmposition.

Hier das kleine Beispiel:

```
IF "<wort1>" EQ "testwort"
SKIP sprung
ENDIF
SKIP fertig
LAB sprung
ECHO "Bedingung erfüllt"
LAB fertig
```

Schließen Sie die IF-Parameter immer in Anführungszeichen ein! Vergessen Sie im EXECUTE-Befehl, den Parameter zu übergeben, lautet unsere erste Zeile:

```
IF "" EQ "testwort"
```

Das ist zwar ein sinnloser Befehl, da unsere Bedingung nie erfüllt wird (ein Leerstring kann niemals ein Wort enthalten!), aber ein zulässiger. Haben Sie die Anführungszeichen nicht eingegeben, lautet die erste Zeile:

```
IF EQ "testwort"
```

darauf hat Amiga-DOS nur eine Antwort:

```
» EQ must have two arguments «
```

(EQ braucht zwei Argumente)

```
» IF failed returncode 20 «
```

(Vergleich nicht möglich, Fehlercode 20)

Im IF-Befehl ist zusätzlich das Wort NOT (nicht) erlaubt. Durch den Zusatz von NOT wird das Abfrageergebnis negiert (aus wahr wird falsch und umgekehrt), das heißt, die Befehle nach IF werden in unserem Beispiel ausgeführt. Zum Beispiel:

```
IF NOT EXISTS "datei"
```

führt die Befehle nach IF aus, wenn eine Datei »datei« nicht existiert. In einer Zeile mit einem IF-Befehl darf neben den Testbedingungen keine weitere Anweisung stehen. Folgende Zeile ist deshalb nicht zulässig:

```
IF "<etwas>" EQ "wahr" SKIP fertig
```

Die korrekte Form lautet:

```
IF "<etwas>" EQ "wahr"
SKIP fertig
ENDIF
...
LAB fertig
```

wobei die Konstante »wahr« nicht in Anführungszeichen stehen muß.

In der nächsten Script-Datei wir die Verschachtelung von IF-Befehlen gezeigt. So können innerhalb einer Befehlsdatei mehrere Bedingungen auf »richtig« oder »falsch« überprüft werden. Tabulatoren erhöhen die Lesbarkeit einer langen Script-Datei ungemein, machen Sie also davon Gebrauch, wenn eine Befehlsdatei öfter verwendet werden und dabei korrigierbar bleiben soll. Die nachfolgende Script-Datei namens »CP« (für »kopiere«) kopiert eine Datei. Als Befehle werden verwendet: IF.....[ELSE]....ENDIF, LAB und SKIP. Die Script-Datei tut nichts anderes als der Befehl COPY, demonstriert aber sinnvolle Strukturen für Befehlsdateien.

```
.KEY von,zu,o          ; Parameter zuordnen
IF "<von>" EQ ""       ; FROM-Datei eingegeben?
SKIP anweisung         ; Keine Datei, dann Befehl
                       ; ausgeben
ENDIF
IF "<zu>" EQ ""        ; TO-Datei eingegeben?
SKIP anweisung         ; Kein File, dann Anweisung
                       ; ausgeben
```

```
ENDIF
IF NOT EXISTS <von> ; FROM-Datei vorhanden?
ECHO "Das Ursprungsfile existiert nicht,"
ECHO "erst mit LIST oder DIR nach der Datei suchen,"
ECHO "dann nochmal starten."
SKIP fertig         ; Es kann jederzeit aus einer
                    ; Schleife herausgesprungen werden.
ENDIF
IF EXISTS <zu>      ; Gibt es die TO-Datei schon?
IF "<o>" EQ "Ü"     ; falls ja, wurde Überschreiben
                    ; eingegeben?
ECHO "Die Datei <zu> wird von einer Kopie der Datei"
ECHO "<von> überschrieben!" ; und die Meldung
                    ; ausgegeben.
ELSE
ECHO "Dieser Befehl überschreibt eine bestehende"
ECHO "Datei nur, wenn der Parameter "Ü" eingegeben wurde"
ECHO "Ausführung abgebrochen!"
SKIP anweisung      ; Sprung zur Anweisung
ENDIF
ELSE
COPY FROM <von> TO <zu>; Datei wird kopiert
                    ; TO-Datei wird neu
                    ; angelegt.
ECHO "Kopiere <von> als <zu>."
ENDIF
SKIP fertig         ; Anweisung soll nur ausgegeben
                    ; werden, wenn ein Fehler gemacht
                    ; wurde.
LAB anweisung
ECHO "Anweisung zum EXECUTE-File CP:"
ECHO "Nur folgende Kopierbefehle werden ausgeführt:"
ECHO "x CP Ursprungsdatei Zieldatei"
ECHO "x CP Ursprungsdatei Zieldatei "
ECHO "wobei x für EXECUTE, CP für diese Befehlsdatei und"
ECHO "für überschreiben einer bestehenden Datei steht."
LAB fertig
```

Jeder einzelne Befehl (auch IF oder ENDIF und so weiter) müssen aus dem C-Verzeichnis geladen werden, bevor Amiga-DOS den Befehl ausführen kann. Bei einer Script-Datei dieser Länge kann das schon eine Weile dauern. Deshalb empfiehlt es sich in vielen Fällen, den C-Ordner oder Teile davon in die RAM-Disk zu kopieren (ASSIGN oder PATH nicht vergessen!), wenn keine Festplatte vorhanden ist, oder oft gebrauchte Befehle resident zu machen.

4.2 DOS-Befehle zur Steuerung von Script-Dateien

ASK

Format: ASK <Eingabeaufforderung>

Muster: ASK "EINGABEAUFFORDERUNG/A"

Zweck: Ermöglicht die Abfrage von Benutzereingaben (ja/nein) bei der Ausführung einer Script-Datei.

Pfad: C:ASK

Beschreibung:

ASK schreibt den Text <Eingabeaufforderung> in den aktuellen Ausgabestrom und wartet dann auf eine Tastatureingabe des Benutzers. Zulässige Anworten sind <Y> (Ja), <N> (Nein) und irgend etwas mit RETURN (identisch mit »Nein«).

Wenn mit »Ja« (Eingabe von Y) geantwortet wurde, wird der Fehlercode auf WARN (=5) gesetzt. Wurde als Antwort "N" eingegeben oder die Taste [Retrun] gedrückt, wird der Fehlercode mit 0 gesetzt. Dieser Befehl ist für den Einsatz in Script-Dateien gedacht.

Siehe auch: IF, EXECUTE

ENDSKIP

Format: ENDSKIP

Muster: ENDSKIP

Zweck: Markierung des Endes eines SKIP-Blocks in einer Befehlsdatei.

Pfad: C:ENDSKIP

Beschreibung:

Wird während der Verarbeitung eines SKIP-Blocks der Befehl ENDSKIP gefunden, wird die Verarbeitung der Script-Datei an der Zeile fortgesetzt, die auf den Befehl ENDSKIP folgt. Der Fehlercode wird auf WARN gesetzt.

EVAL

Format: EVAL <Wert1> <Operation> <Wert2> [TO <Datei>] [LFORMAT= <Zeichenfolge>]

Schablone: EVAL "WERT1,OPERATION,WERT2,TO/K,LFORMAT/K"

Zweck: Der Wert einfacher Ausdrücke wird ermittelt.

Pfad: C:EVAL

Beschreibung:

EVAL läßt sich zum Berechnen und Ausgeben des Ergebnisses einfacher Ganzzahl-Ausdrücke (mit 1 und 2 Argumenten) verwenden. <Wert1> und <Wert2> können im dezimalen, hexadezimalen oder oktalen Zahlensystem angegeben werden. Dezimalzahlen werden als Standard angenommen. Hexadezimalzahlen werden durch ein führendes 0X oder #X gekennzeichnet. Oktalzahlen mit führender »0« (Null) oder »#« gekennzeichnet (gefolgt von den Ziffern der Zahl). Buchstaben werden durch ein einzelnes Hochkomma (') (Alt+ä) gekennzeichnet.

Folgende Operationen lassen sich mit EVAL ausführen:

Addition	+
Subtraktion	-
Multiplikation	*
Division	/
Modulo	mod
Und	&
Oder	|
Nicht	~
Verschiebung nach links	<<
Verschiebung nach rechts	>>
Negation	-
Exklusiv Oder	xor
Bitweise Identität	eqv

Die Ausgabe erfolgt normalerweise als Dezimalzahl, durch das Schlüsselwort LFORMAT kann das Format der Darstellung selbst gewählt werden. Als Formatanweisungen können %X (hexadezimal), %O (oktal), %N (dezimal) oder %C (Zeichen) verwendet werden. Bei den Optionen % X und %O muß die Anzahl der auszugegebenden Stellen der Formatsequenz nachgestellt werden (d.h. %O4 bewirkt eine Ausgabe von vier Stellen einer Oktalzahl).

Mit LFORMAT kann man auch unter Angabe von »*N« einen Zeilenvorschub in der Ausgabe bewirken.

EVAL kann in Script-Dateien zum Zählen von Schleifendurchläufen verwendet werden. In diesem Fall ist die Option TO, welche die Ausgabe von EVAL zu einer Datei sendet, sehr hilfreich.

Beispiele:

```
EVAL 3 * 27
```

ergibt

```
81
EVAL 22 - 11 LFORMAT="Das Ergebnis ist %N4*N")
```

druckt

```
Das Ergebnis ist 11
```

FAILAT

Format: FAILAT <n>

Muster: FAILAT "RETURNCODELIMIT"

Zweck: Verändern des Abbruchlimits für Script-Dateien bei Fehlern.

Pfad: C:FAILAT

Beschreibung:

FAILAT setzt die Schranke für den Rückgabewert, den eine DOS-Anweisung beim Auftreten eines Fehlers zurückgibt, um die Bearbeitung der Script-Datei abzubrechen.

Befehle, die nicht ausgeführt werden konnten, zeigen dies an, indem sie einen Rückgabewert (Returncode) zurückmelden. Ein Rückgabewert größer als null (0) zeigt an, daß ein Fehler eingetreten ist. Ein Fehlercode größer oder gleich einem bestimmten Wert (der sogenannten Fehlerschranke) bricht eine Folge nicht interaktiver Befehle ab (zum Beispiel in einer Script-Datei oder die Kommandos, die Sie nach einem RUN-Befehl eingeben).

Der Fehlercode zeigt dabei an, wie schwerwiegend der Fehler war. Er nimmt in der Regel die Werte 5 (WARN), 10 (ERROR) oder 20 (FAIL) an. Der Befehl FAILAT wird nun benutzt, um die Abbruchschranke, deren Wert auf 10 voreingestellt ist, zu verändern. Wird der Wert vergrößert, werden die Fehler, die um eins kleiner sind als die Abbruchschranke akzeptiert und die Ausführung nachfolgender Befehle wird trotz Fehler fortgesetzt.

FAILAT kann in Befehlsfolgen verwendet werden, um nach Befehlen testen zu können, ob eine Fehlerbedingung auftrat. Falls ein Fehler auftrat kann dann in der Script-Datei eine Fehlerbehandlung ausgeführt werden, um dem Anwender der Script-Datei ein komfortables Werkzeug in die Hand zu geben.

Wird das Argument weggelassen, zeigt FAILAT den augenblicklich eingestellten Wert an.

Beispiel(e):

Die nachfolgende Befehlssequenz wird nur abgebrochen, wenn ein Fehler-Return-Code >= 25 auftritt.

```
FAILAT 25
```

ICONX

Format: ICONX

Muster: ICONX

Zweck: Ausführung einer Amiga-DOS-Script-Datei von der Workbench aus (Icon anklicken).

Pfad: C:ICONX

Beschreibung:

ICONX erlaubt Script-Dateien von der Workbench ausführen zu lassen. ICONX wechselt das aktuelle Verzeichnis zu dem Verzeichnis, das das Projekt-Icon der Script-Datei enthält, bevor es die Befehlsdatei ausführt.

Nach dem Erstellen einer Script-Datei mit einem Texteditor (z.B. Micro-Emacs) muß der Datei ein Project-Icon zugeordnet werden, indem man es in »Dateiname.info« umbenennt und in das gleiche Verzeichnis wie die Datei kopiert, wenn nicht gleich vom Texteditor die Datei mit Projekt-Icon gespeichert wird.

Danach ändert man das »DEFAULT TOOL« des Icons zu »C:ICONX« und speichert mit SAVE das Icon wieder. Durch Öffnen (Doppelklick) wird die Script-Datei ausgeführt. Dafür wird auf der Workbench ein Console-Fenster geöffnet; in dem Feld »TOOL TYPES« des Icons kann die Größe des Fensters eingetragen werden:

```
WINDOW=CON:0/12/704/450/"ziemlich groß"
```

Durch die Angabe von

```
DELAY=nnn
```

in den TOOL TYPES wird eine kurze Pause nach der Abarbeitung der Datei bewirkt, die es ermöglicht, einen ausgegebenen Text zu lesen. Wird eine Null (0) für nnn angegeben, wartet ICONX auf Control-C als Befehl zum Beenden.

Die erweiterte Auswahl der Workbench kann verwendet werden, um Dateien mit Icons an die Befehlsdatei zu übergeben. Diese Dateien werden der Befehlsdatei als Argumente übergeben, falls diese mit ».KEY« die Eingabe von Argumenten verlangt. ».KEY""« oder »[Leertaste]« erlaubt den Einsatz von Befehlen, die eine Eingabe in eine ICONX-Befehlsdatei verlangen.

Siehe auch: EXECUTE

IF

Format: IF [NOT] [WARN] [ERROR] [FAIL] [<Zeichenkette> EQ|GT|GE <Zeichenkette>] [VAL] [EXISTS<Dateiname>]

Muster: IF "NOT/S, WARN/S, ERROR/S, FAIL/S, EQ/K, GT/K, GE/K, VAL/S, EXISTS/K"

Zweck: Abfragen einer Bedingung in Script-Dateien.

Pfad: C:IF

Beschreibung:

IF erlaubt die Abfrage von Bedingungen in Script-Dateien. Dieser Befehl kann nur in einer Script-Datei sinnvoll verwendet werden.

Sind eine oder mehrere Bedingungen erfüllt, führt IF die folgenden Befehle aus, bis er einen Befehl ELSE oder ENDIF findet. Sind andernfalls die Bedingungen nicht erfüllt,

führt Amiga-DOS alles aus, was hinter dem entsprechenden Befehl ELSE folgt. ENDIF und ELSE sind ebenfalls nur in Script-Dateien sinnvoll die IF enthalten.

ENDIF beendet einen IF-Befehl. Wird die IF-Bedingung nicht erfüllt, gibt ELSE den weiteren Ablauf des Befehls an.

IF-Anweisungen können verschachtelt werden, dabei ist nur zu beachten, daß jedes IF mit einem ENDIF abgeschlossen wird.

Zulässig ist zum Beispiel folgende Konstruktion:

```
IF <Bedingung>
 IF <Bedingung>
  IF <Bedingung>
   Befehl
   Befehl
   Befehl
  ENDIF
  ELSE
   IF <Bedingung>
    Befehl
    Befehl
   ENDIF
 ENDIF
ENDIF
```

Beachten Sie bitte, daß ELSE nach Bedarf eingesetzt werden kann und daß verschachtelte IFs zum nächstgelegenen ENDIF springen.

Die Konstante ERROR kann nur dann abgefragt werden, wenn FAILAT größer 10 gesetzt wurde. In gleicher Weise ist FAIL nur verfügbar, wenn FAILAT größer 20 gesetzt wurde, da ansonsten der Kommandozeilen-Interpreter die Abarbeitung von Script-Dateien bei Auftreten des Rückgabewertes ERROR beziehungsweise FAIL abbricht.

Die Schlüsselwerte haben folgende Funktion:

Schlüsselwort	Funktion
NOT	negiert das Ergebnis
WARN	erfüllt, wenn der letzte Rückgabewert größer oder gleich 5 ist.
ERROR	erfüllt, wenn der letzte Rückgabewert größer oder gleich 10 ist.
FAIL	erfüllt, wenn der letzte Rückgabewert größer oder gleich 20 ist.
<a> EQ <b>	erfüllt, wenn a und b identisch sind.
<a> GE <b>	erfüllt, wenn a größer oder gleich b ist.
<a> GT <b>	erfüllt, wenn a größer als b ist.

Vergleiche werden als Vergleiche von Zeichenfolgen durchgeführt, wenn nicht die Option VAL angegeben wird, dann erfolgt ein numerischer Vergleich.

EXISTS überprüft, ob eine Datei, ein Verzeichnis oder ein logisches Gerät vorhanden ist:

```
IF EXISTS <Datei>
```

ist erfüllt, wenn <Datei> existiert.

Mit IF können Umgebungsvariablen verglichen werden. Soll eine Umgebungsvariable abgefragt werden, so muß den Namen der Umgebungsvariablen ein Dollar-Zeichen »$« vorangestellt werden.

IF EQ kann mit folgender Form dazu verwendet werden, einen ungesetzten Parameter in einer Befehlssequenz zu entdecken:

```
IF "<a>" EQ ""
 ECHO "Parameter nicht definiert!"
 SKIP ende
ENDIF
...
LAB ende
```

Durch Kombination können die Vergleiche »kleiner oder gleich« NOT GT oder »kleiner als« NOT GE konstruiert werden.

Beispiel:

```
IF EXISTS Arbeit/Hugo
 TYPE Arbeit/Hugo
ELSE
 ECHO "Hugo nicht gefunden!"
ENDIF
```

Gibt es die Datei »Hugo« im Verzeichnis »Arbeit«, wird sie auf dem Bildschirm ausgegeben. Gibt es die Datei nicht, erscheint die Fehlermeldung »Hugo nicht gefunden!« und der nächste Befehl wird ausgeführt.

```
FAILAT 11
IF ERROR
 SKIP fehlerlab
ENDIF
...
LAB fehlerlab
```

Verzweigt zum (vorher erstellten) LAB fehlerlab, wenn ein Fehler größer gleich Fehlercode 10 aufgetreten ist.

```
IF ERROR
 IF EXISTS fred
  ECHO "»fred« ex., aber es wurde vorher ein Fehler entdeckt"
 ENDIF
ENDIF
```

LAB

Format: LAB <Sprungmarke>

Muster: LAB "SPRUNGMARKE/A"

Zweck: Sprungziele definieren, die mit SKIP angesprungen werden.

Pfad: C:LAB

Beschreibung:

LAB kennzeichnet Sprungziele in Script-Dateien. Mit LAB wird ein Sprungziel definiert, das mit dem Befehl SKIP angesprungen werden kann.

Beispiel(e):

```
LAB ende
```

definiert das Sprungziel »ende«, das mit

```
SKIP ende
```

angesprungen wird.

QUIT

Format: QUIT [<returncode>]

Muster: QUIT "RC"

Zweck: Eine Befehlssequenz abbrechen und Fehlercode zurückgeben.

Beschreibung:

QUIT unterbricht eine Befehlssequenz und gibt dem Benutzer eine Fehlernummer aus.

Beispiel:

```
QUIT
```

Der Befehl unterbricht die Befehlssequenz ohne Fehlermeldung.

```
FAILAT 30
IF ERROR
  QUIT 20
ENDIF
```

Ist beim letzten Kommando ein Fehler aufgetaucht, dann unterbricht diese Sequenz mit der Fehlernummer 20.

SKIP

Format: SKIP <label> [BACK]

Muster: SKIP "LABEL, BACK/S"

Zweck: Sprung zu einem definierten Label in einer Befehls-Sequenz.

Beschreibung:

SKIP führt einen Sprung in einer Script-Datei aus. Der Befehl SKIP wird in Verbindung mit LAB verwendet.

SKIP kann mit oder ohne Label verwendet werden. Ohne Label-Eingabe findet SKIP den nächsten unbenannten LAB-Befehl. Mit der Angabe eines Labels versucht Amiga-DOS, einen LAB-Befehl mit dem gleichen Labelnamen zu finden. LAB muß das erste Wort in einer Zeile der Datei sein. Wenn SKIP das bezeichnete Label nicht finden kann, wird die Befehlsfolge beendet und Amiga-DOS gibt folgende Meldung aus:

```
»label »name« not found by Skip«
```

Wird das Schlüsselwort BACK angegeben, beginnt SKIP mit der Suche nach dem entsprechenden LABEL nach dem, dem SKIP-Befehl vorangehenden EXECUTE. Ist kein EXECUTE in der Script-Datei vorhanden, beginnt SKIP mit der Suche am Anfang der Datei. Mit ENDSKIP kann die Suche abgebrochen werden.

Beispiel(e):

```
SKIP
```

springt zum nächsten Label, welches keinen Namen trägt.

```
IF ERROR
  SKIP fehlerbearbeitung
ENDIF
```

springt zum Label »fehlerbearbeitung«, wenn in den vorhergehenden Befehlen ein Fehler aufgetreten ist. Eine solche oder ähnliche Sequenz kann zum Beispiel wie im folgenden Beispiel verwendet werden:

```
FAILAT 100
ASSEM text
IF ERROR
SKIP fehler
ENDIF
LINKSKIP fertig
LAB fehler
ECHO "Fehler bei ASSEM"
LAB fertig
ECHO "Nächster Befehl bitte"
```

4.3 Ein Spezialfall: Die Startup-Sequence

Die Startup-Sequence ist eine Script-Datei, mit der jeder Amiga-Anwender zwangsweise bei jedem Reboot arbeitet. Das Amiga-DOS führt dann nämlich immer die Datei »Startup-Sequence« im Verzeichnis »SYS:S« aus, um bestimmte Voreinstellungen zu treffen, die

für das Arbeiten mit dem Amiga notwendig sind, um die Arbeitsumgebung des Benutzers einzurichten.

Die Startup-Sequence der Original-Workbench1.3D sieht unverändert so aus:

```
Addbuffers df0: 10       ;Pufferspeicher für df0: vergrößern
c:SetPatch >NIL:         ;Systemroutinen korrigieren
cd c:                    ;Aktuelles Verzeichnis wird SYS:C
echo "A500/A2000 Workbench disk. Release 1.3 Version 34.20*N"
Sys:System/FastMemFirst
  ;SLOW-MEMORY ans Ende der Systemspeicherliste setzen
BindDrivers
  ;Gerätetreiber aus dem EXPANSION-Verzeichnis laden
SetClock load
  ;Systemuhrzeit aus batteriegepufferter Echtzeituhr laden.
  ;Wenn keine Echtzeituhr vorhanden ist, sollte SetClock durch
  ;»Date ?« ersetzt werden.
FF >NIL: -0
  ;Textausgabe in Fenstern beschleunigen
resident CLI L:Shell-Seg SYSTEM pure add
  ;SHELL anstatt CLI im System installieren
resident c:Execute pure ;EXECUTE resident machen
mount newcon:            ;NEWCON: installieren
;
failat 11
  ;Abbruchschranke erhöhen, so daß ERROR abgefragt werden kann.
run execute s:StartupII
  ;DOS-Befehle resident machen, weitere logische Verzeichnisse
  ;setzen und Handler anmelden.
wait >NIL: 5 mins        ;Ende von StartupII abwarten
;
SYS:System/SetMap d
  ;Deutsche Tastaturbelegung laden
path ram: c: sys:utilities sys:system s: sys:prefs add
  ;Suchpfad für DOS-Befehle festlegen
LoadWB delay             ;Workbench starten
endcli >NIL:             ;CLI-Fenster schließen
Die Datei StartupII hat folgenden Inhalt:
resident c:Resident pure
resident c:List pure
resident c:CD pure
resident c:Mount pure
resident c:Assign pure
```

```
resident c:Makedir pure
  ;ausgesuchte DOS-Befehle resident machen
  ;IF, ENDIF, ELSE, SKIP, ENDSKIP und ECHO resident, falls
  ;viele Scripts benutzt werden und ausreichend Speicher
  ;vorhanden ist.
  ;Failat, WAIT und ENDCLI ebenfalls resident machen, wenn
  ;IconX oft benutzt wird.
makedir ram:t
assign T: ram:t          ;T:-Verzeichnis für Scripts
makedir ram:env
assign ENV: ram:env
  ;ENV:-Verzeichnis für Umgebungsvariablen anlegen.
makedir ram:clipboards
assign CLIPS: ram:clipboards
  ;CLIPS: für den CLIPS-Handler anlegen (der Handler
  ;funktioniert unter 1.3 noch nicht richtig.
mount speak:
mount aux:
mount pipe:
  ;nützliche Handler anmelden
resident Mount remove
resident Assign remove
resident Makedir remove
  ;Wenn genug Speicher vorhanden resident lassen, indem die
  ;obigen drei Zeilen auskommentiert werden (Vorstellen eines
  ;Semikolons (;).
;
break 1 C
  ;Signal an ersten SHELL-Prozeß, in dem die Startup-Sequence
  ;ausgeführt wird, daß er mit der Ausführung des Rests der
  ;Startup-Sequence fortfahren kann.
```

Die Bedeutung der einzelnen DOS-Befehle läßt sich im Kapitel 3, der alphabetischen Erklärung dieser Befehle, für eine detaillierte Erläuterung nachschlagen, falls die Kurzkommentare zu jeder Zeile zu knapp ausgefallen sind.

Die originale Startup-Sequence kann natürlich nicht für jede Arbeitsumgebung geeignet sein und stellt den Versuch dar, jedem Anwender eine erste Annäherung an seine ideale Startup-Sequence zu bieten. So können zum Beispiel wesentlich mehr Befehle resident gemacht werden, wenn im Rechner eine Speichererweiterung eingebaut ist. Utilities wie CLOCKPOINTER können gestartet werden, eine SHELL kann automatisch geöffnet werden, zusätzliche Pfade und logische Geräte können angemeldet werden und eine Festplatte oder die RAD: können in das Dateisystem eingebunden werden. Falls einige Leser schon ein Kickstart-ROM der Version 1.3 besitzen, können sie auch schon von der RAD: oder

ihrer Festplatte booten, und wollen daher ihre Startup-Sequence stark ändern, um diese Möglichkeit zu nutzen.

Machen Sie sich also mit einem Texteditor vertraut, und laden Sie die Datei S:Startup-Sequence, um sie an Ihre persönlichen Bedürfnisse und Wünsche anzupassen.

4.3.1 Eine eigene Startup-Sequence

Folgende Startup-Sequence soll ein übersichtliches Gerüst für eigene Veränderungen bieten:

```
SYS:C/SETPATCH >NIL:
SYS:C/CD SYS:C
SYS:SYSTEM/FASTMEMFIRST
BINDDRIVERS
SETCLOCK >NIL: LOAD
SYS:SYSTEM/SETMAP D
FF >NIL: -O
RESIDENT SYS:C/EXECUTE PURE
EXECUTE SYS:S/MAKERESIDENT
EXECUTE SYS:S/MAKEMOUNT
EXECUTE SYS:S/MAKEASSIGN
PATH S: SYS:PREFS SYS:UTILITIES SYSTEM: C:
LOADWB
NEWSHELL NEWCON:0/12/640/240/AmigaSHELL
ENDCLI >NIL:
```

Die Dateien »MAKERESIDENT«, »MAKEASSIGN« und »MAKEMOUNT« müssen ebenfalls noch erstellt werden:

```
MAKERESIDENT:
RESIDNET CLI L:SHELL-SEG SYSTEM PURE ADD
RESIDENT SYS:C/RESIDENT
RESIDENT SYS:C/LIST
RESIDENT SYS:C/CD
RESIDENT SYS:C/MOUNT
RESIDENT SYS:C/ASSIGN
RESIDENT SYS:C/MAKEDIR
RESIDENT SYS:C/IF
RESIDENT SYS:C/ENDIF
RESIDENT SYS:C/ELSE
RESIDENT SYS:C/SKIP
RESIDENT SYS:C/ENDSKIP
RESIDENT SYS:C/ECHO
RESIDENT SYS:C/FAILAT
```

```
RESIDENT SYS:C/WAIT
RESIDENT SYS:C/NEWSHELL
RESIDENT SYS:C/ENDCLI
```

Diese Datei kann beliebig ergänzt oder gekürtzt werden, wobei nur zu beachten ist, daß Befehle, die resident gemacht werden sollen, das »P«-Schutzflag gesetzt haben, das der LIST-Befehl anzeigt (siehe auch DOS-Befehl RESIDENT). Besitzer einer schnellen Festplatte werden gerne darauf verzichten, Befehle resident zu machen, da sich die Ladezeiten von DOS-Befehlen dank der Festplatte auf ein kaum merkliches Minimum beschränken, und können somit ganz auf den Aufruf von MAKERESIDENT verzichten.

```
MAKEASSIGN:
ASSIGN C:                SYS:C
ASSIGN S:                SYS:S
ASSIGN L:                SYS:l
ASSIGN LIBS:             SYS:LIBS
ASSIGN DEVS:             SYS:DEVS
ASSIGN FONTS:            SYS:FONTS/SYS
ASSIGN SYSTEM:           SYS:SYSTEM
ASSIGN ENV:              S:ENV
MAKEDIR RAM:T
ASSIGN T:                RAM:T
MAKEDIR RAM:CLIPBOARDS
ASSIGN CLIPS:            RAM:CLIPBOARDS
  ;private Vereinbarungen
ASSIGN WORK:             USR:ADOSHB_Manusscript
  ;für die Freunde des Lattice-C-Compilers
ASSIGN LC:               SYS:LC/C
ASSIGN LIB:              SYS:LC/LIB
ASSIGN INCLUDE:          SYS:LC/INCLUDE
ASSIGN QUAD:             T:
```

Mit Hilfe von ASSIGN werden logische Geräte eingerichtet, auf die direkt durch Angabe ihres Namens zugegriffen werden kann. Das Amiga-DOS benötigt alle Vereinbarungen bis zum Teil mit den privaten, individuell verschiedenen Vereinbarungen und legt die entsprechenden Geräte selbstständig an, wenn der Benutzer darauf verzichtet.

Mit dem logischen Verzeichnis WORK: erreicht man schnell seine Arbeiten für ein aktuelles Projekt, und Benutzer eines Lattice-C-Compilers benötigen die letzten vier Vereinbarungen, damit ihr Compiler korrekt arbeiten kann.

Wenn das logische Gerät ENV: als Unterverzeichnis zum Beispiel von S: auf einem Speichermedium angelegt wird, bleibt der Inhalt aller Umgebungsvariablen erhalten und muß nicht nach einem Reboot neu gesetzt werden (siehe auch SETENV, GETENV).

Mit der Script-Datei MAKEMOUNT sollten zuvor noch alle Geräte angemeldet werden, die Sie bei Ihrer Arbeit brauchen und auf die vielleicht ein logisches Gerät verweisen soll:

```
MAKEMOUNT:
MOUNT NEWCON:          ;!!! SHELL
MOUNT PIPE:
MOUNT AUX:
MOUNT RAD:
MOUNT DH1:
MOUNT DH2:
MOUNT ETHERNET:
MOUNT OPTICAL:
```

Die beiden letzten Zeilen sollen darauf hindeuten, daß der Amiga auch das Betreiben von heute noch wenig verbreiteten Geräten wie Netzen auf Basis von »Ethernet« oder den neuentwickelten thermo-magneto-optischen Disketten mit phantastischen Speicherkapazitäten erlauben kann.

Es ist denkbar, daß der Amiga auch von einem Netzwerk oder einer Festplatte bootet, wenn der Festplattencontroller (Commodore A2090A oder A2090 mit zusätzlicher Steckkarte A2090B bootet vom alten Filing-System, Microbotics SCSI-Kontroller und Great-Valley-Products SCSI-Kontroller Booten vom Fast-Filing-System) oder der Netzwerkadapter autokonfigurierende Erweiterungskarten sind, auf denen ROM-(Festwert-)speicher vorhanden sind, die das Autoboot in Verbindung mit Kickstart 1.3 gestatten.

Genaue Informationen über den Befehl MOUNT und über die Einträge in die Mountlist für Festplattenpartitionen können dem vorherigen Kapitel über MOUNT und der Datei DEVS:Mountlist entnommen werden.

4.3.2 Booten vom Fast-Filing-System

Für Besitzer einer Autoboot-Festplatte, die mit einem Controller von »Microbotics« oder »Great-Valley-Products« betrieben wird, und des Kickstart 1.3 ist die Anpassung der Startup-Sequence an ihre Verhältnisse sehr einfach. Die Festplatte wird, wie in der jeweiligen Produktanleitung beschrieben, partitioniert und dann für das Fast-Filing-System formatiert. Diese Festplatten brauchen keinen Eintrag in der Datei »DEVS:Mountlist«, denn beim Einrichten der Festplatte werden auf deren Spur Null eine Art Mountlisteintrag und das Fast-Filing-System kopiert. Ein spezieller Gerätetreiber braucht auch nicht in den EXPANSION-Ordner kopiert werden, da die Treibersoftware schon im ROM auf dem Festplattencontroller vorhanden ist.

Die Platte wird beim Reboot automatisch erkannt, und das System benutzt die erste Partition wie eine Workbench-Diskette zum Booten. Deshalb sollte der Inhalt der Workbench1.3D-Diskette möglichst komplett auf die Festplatte kopiert werden, damit die Workbenchdiskette nicht mehr gebraucht wird, und die obige Startup-Sequence muß nur noch für all die Programme um Einträge erweitert werden, die von der Festplatte benutzt werden sollen, und spezielle logische Geräte, oder bestimmte Vorbereitungen benötigen.

4.3.3 Einrichten einer Bootpartition für A2090A

Wenn Sie einen Commodore A2090A-Controller besitzen, wird es etwas schwieriger, da der Controller nur von Partitionen, die mit dem alten Filing-System formatiert sind, und nicht von Fast-Filing-System-Partitionen booten kann. Deshalb muß auf der Festplatte die erste Partition DH0: mit dem alten Filing-System formatiert werden. Diese Partition braucht aber nicht größer als die Speicherkapazität einer normalen Diskette zu sein, da beim Bootvorgang die Kontrolle an weitere Partitionen unter Fast-Filing-System abgegeben werden kann.

Minimal sollte die Größe der DH0:-Partition zirka 150 Kbyte betragen, um alle zum Reboot notwendigen Dateien zu enthalten, und sollte folgenden Inhalt haben:

```
c     (dir)
      binddrivers                    setpatch
      mount
l     (dir)
      Aux-Handler                    Disk-Validator
      Fast-File-System                 Newcon-Handler
      Pipe-Handler                   Port-Handler
      Ram-Handler                    Shell-Seg
      Speak-Handler
devs  (dir)
      mountlist                      system-configuration
s (dir)
      startup-sequence
libs  (dir)
      icon.library
.info                                Disk.info
```

Tabelle 4.1: Die Files für die DHO:-Partition

Mindestinhalt einer Bootpartition, Diskette oder RAD:

Dann sollte die Datei »S:Startup-Sequence« auf der Partition DH0: folgendes Aussehen haben:

```
SYS:C/SETPATCH >NIL:
SYS:C/BINDDRIVERS
  ;Einbinden von nicht autobootfähigen Geräten
SYS:C/MOUNT DH1:
  ;Fast-File-System Partition
DH1:SYSTEM/FASTMEMFIRST
DH1:C/CD DH1:C
ASSIGN SYS: DH1:
SETCLOCK >NIL: LOAD
```

```
SYS:SYSTEM/SETMAP D
FF >NIL: -0
RESIDENT SYS:C/EXECUTE PURE
EXECUTE SYS:S/MAKERESIDENT
EXECUTE SYS:S/MAKEMOUNT
EXECUTE SYS:S/MAKEASSIGN
PATH S: SYS:PREFS SYS:UTILITIES SYSTEM: C:
LOADWB
NEWSHELL NEWCON:0/12/640/240/AmigaSHELL
ENDCLI >NIL:
```

Die drei Script-Dateien MAKERESIDENT, MAKEMOUNT UND MAKEASSIGN brauchen nicht geändert werden (DH1: braucht nicht ein zweites Mal »gemounted« werden).

4.3.4 Booten einer nicht autobootfähigen Festplatte von Diskette

Wenn kein autobootfähiger Controller oder kein Kickstart 1.3 zur Verfügung steht, muß das System von Diskette gebootet werden. Dazu muß auf der Bootdiskette der gleiche Minimalinhalt vorhanden sein, wie auf der oben beschriebenen Bootpartition DH0: bei der Verwendung eines Controllers, der nicht vom Fast-Filing-System booten kann. Nur muß außerdem im Verzeichnis EXPANSION auf der Diskette der Gerätetreiber für den Festplattencontroller vorhanden sein.

Die »Startup-Sequence« muß gegenüber dem vorherigen Beispiel nur minimal verändert werden:

```
SYS:C/SETPATCH >NIL:
SYS:C/BINDDRIVERS
  ;Einbiden von nicht autobootfähigen Geräten
SYS:C/MOUNT DH0:
  ;erste Fast-File-System Partition
DH0:SYSTEM/FASTMEMFIRST
DH0:C/CD DH0:C
ASSIGN SYS: DH0:
SETCLOCK >NIL: LOAD
SYS:SYSTEM/SETMAP D
FF >NIL: -0
RESIDENT SYS:C/EXECUTE PURE
EXECUTE SYS:S/MAKERESIDENT
EXECUTE SYS:S/MAKEMOUNT
EXECUTE SYS:S/MAKEASSIGN
PATH S: SYS:PREFS SYS:UTILITIES SYSTEM: C:
LOADWB
NEWSHELL NEWCON:0/12/640/240/Amiga-SHELL
ENDCLI >NIL:
```

Mit einer kleinen Script-Datei wird eine vollständige Boot-Diskette erzeugt:

```
MAKEBOOTDISK:
ECHO "Legen Sie bitte eine *e[42m*e[33mleere Diskette *e[40*e[30min
      das"
ECHO "Laufwerk DF0:, die *e[42m*e[33mformatiert *e[40m*e[30mwerden
kann!"
ECHO "Die Disk wird jetzt formatiert,"
ECHO "sie ist nicht schreibgeschützt."
ASK  "Null Problemo ! (*e[1mY*e[0m(ja) / *e[1mN*e[0mein)"
IF NOT WARN
  ECHO "Dann können wir nicht weiter machen !"
  QUIT ERROR
ENDIF
ECHO "Drücken Sie bitte wenn angefordert *e[1<RETURN>*e[0."
FORMAT DRIVE DF0: NAME "BOOTWB" NOICONS
ECHO "Jetzt werden alle erforderlichen Dateien kopiert."
MAKEDIR DF0:S
MAKEDIR DF0:DEVS
MAKEDIR DF0:EXPANSION
MAKEDIR DF0:LIBS
MAKEDIR DF0:L
MAKEDIR DF0:C
COPY DEVS:SYSTEM-CONFIGURATION DF0:DEVS
COPY DEVS:MOUNTLIST DF0:DEVS
COPY SYS:EXPANSION DF0:EXPANSION QUIET
COPY LIBS:ICON.LIBRARY DF0:LIBS
COPY L: DF0:L QUIET
COPY S:STARTUP-SEQUENCE DF0:S
ECHO "[Ctrl]+[G], und schon sind wir fertig!, Viel Spaß beim
      Arbeiten!"
```

Dabei muß natürlich darauf geachtet werden, daß auf der Bootdiskette SYS: im EXPANSION-Ordner alle notwendigen Gerätetreiber vorhanden sind.

4.3.5 Booten von der RAD:

Wenn der Festplattencontroller nicht booten kann, aber ein Kickstart 1.3 ROM vorhanden ist, kann auch die RAD: zum Booten verwendet werden. Die RAD (Recoverable RAM Disk) ist eine Diskettenstation, die im RAM simuliert wird, beim Reboot aber nicht wie die RAM:-Disk ihren Inhalt verliert.

Die einfachste Möglichkeit von der RAD: zu booten ist, wie im Kapitel über die Mountlist beschrieben, die Größe der RAD: so einzustellen, daß sie genauso groß ist, wie eine Diskette, und dann eine bootfähige Diskette mit

```
DISKCOPY FROM DF0: TO RAD:
```

in die RAD: zu kopieren. Um eine ganze Diskette aufzunehmen, benötigt die RAD: jedoch ein Mbyte Speicherplatz im RAM des Amiga. Aber auch in diesem Fall kann die RAD: auf zirka 150 Kbyte Minimalgröße verkleinert werden, falls RAM gespart werden soll oder muß, um die oben aufgeführten Dateien aufzunehmen, die für das Booten minimal erforderlich sind. Das ist interessant, wenn Sie den alten Commodore A2090 Festplattencontroller besitzen, der nicht autobootfähig ist, um dennoch ohne Diskette im Laufwerk df0: zu booten.

Startup-Sequence auf Boot-Diskette:

```
SYS:C/SETPATCH >NIL:
SYS:C/BINDDRIVERS
  ;Einbiden von nicht autobootfähigen Geräten
SYS:C/MOUNT DH0:
  ;erste Fast-File-System Partition
DH0:SYSTEM/FASTMEMFIRST
DH0:C/CD DH0:C
ASSIGN SYS: DH0:
SETCLOCK >NIL: LOAD
SYS:SYSTEM/SETMAP D
FF >NIL: -0
RESIDENT SYS:C/EXECUTE PURE
EXECUTE SYS:S/MAKERESIDENT
EXECUTE SYS:S/MAKEMOUNT
EXECUTE SYS:S/MAKEASSIGN
FAILAT 31
ASSIGN >NIL: RAD: EXISTS
IF WARN
  ECHO "MOUNT RAD:"
  MOUNT RAD:
  EXECUTE SYS:S/MAKEBOOTRAD
ENDIF
FAILAT 10
PATH S: SYS:PREFS SYS:UTILITIES SYSTEM: C:
LOADWB
NEWSHELL NEWCON:0/12/640/240/AmigaSHELL
ENDCLI >NIL:
```

Nach einem Kaltstart (dem Einschalten des Rechners) muß der Amiga zuerst normal mit einer Workbench-Diskette gebootet werden. Doch während des Bootens wird die RAD: installiert und mit allen nötigen Daten bespielt.

```
MAKEBOOTRAD:
MAKEDIR RAD:S
MAKEDIR RAD:DEVS
MAKEDIR RAD:EXPANSION
MAKEDIR RAD:LIBS
MAKEDIR RAD:L
MAKEDIR RAD:C
COPY DEVS:SYSTEM-CONFIGURATION RAD:DEVS
COPY DEVS:MOUNTLIST RAD:DEVS
COPY SYS:EXPANSION RAD:EXPANSION QUIET
COPY LIBS:ICON.LIBRARY RAD:LIBS
COPY L: RAD:L QUIET
COPY S:STARTUP-SEQUENCE RAD:S
```

MAKEBOOTRAD ist natürlich eine verkürzte und speziell angepaßte Version von MAKEBOOTDISK.

4.3.6 Anlegen einer nicht bootfähigen RAD

Sollte nur der Kickstart 1.2 ohne Autoboot zur Verfügung stehen, kann die RAD: trotzdem sinnvoll genutzt werden:

```
SYS:C/SETPATCH >NIL:
SYS:C/BINDDRIVERS
  ;Einbiden von nicht autobootfähigen Geräten
SYS:SYSTEM/FASTMEMFIRST
FAILAT 31
ASSIGN >NIL: RAD: EXISTS
IF WARN
  ECHO "MOUNT RAD:"
  MOUNT RAD:
  EXECUTE WB2RAD
ENDIF
FAILAT 10
SYS:C/CD RAD:C
ASSIGN SYS: RAD:
SETCLOCK >NIL: LOAD
SYS:SYSTEM/SETMAP D
FF >NIL: -O
RESIDENT SYS:C/EXECUTE PURE
EXECUTE SYS:S/MAKERESIDENT ;entfällt meist, wenn C: auf RAD:
EXECUTE SYS:S/MAKEMOUNT
EXECUTE SYS:S/MAKEASSIGN
PATH S: SYS:PREFS SYS:UTILITIES SYSTEM: C:
```

```
LOADWB
NEWSHELL NEWCON:0/12/640/240/Amiga-SHELL
ENDCLI >NIL:
```

Jetzt fehlt nur noch

```
WB2RAD:
MAKEDIR RAD:S
MAKEDIR RAD:DEVS
MAKEDIR RAD:PRINTERS
MAKEDIR RAD:LIBS
MAKEDIR RAD:L
MAKEDIR RAD:C
COPY DEVS:MOUNTLIST RAD:DEVS
COPY DEVS:#? RAD:DEVS QUIET
COPY DEVS:PRINTERS/»mein Treiber« RAD:DEVS/PRINTERS
COPY LIBS: RAD:LIBS QUIET
COPY L: RAD:L QUIET
COPY S: RAD:S ALL QUIET
```

Durch diese »Startup-Sequence« wird die RAD: nach dem Kaltstart einmal mit wichtigen System-Dateien beschrieben und das System auf die RAD: umgelenkt. Wenn ein Anwender keine Festplatte, aber sehr viel Speicher hat, kann es auch sehr nützlich sein zum Beispiel den C-Compiler mit allen benötigten INCLUDE-Dateien und LIBRARIES auf die RAD: zu kopieren. Die RAD: überlebt meistens auch Abstürze, die durch fehlerhafte Programme verursacht werden.

4.4 Patternmatching mit SPAT und DPAT

SPAT

Format: SPAT <Befehl> <PATTERN> [OPTIONS]*

Zweck: Patternmatching für DOS-Befehle, die kein Patternmatching (Suchmuster siehe auch LIST) direkt unterstützen.

Pfad: S:SPAT

Beschreibung:

SPAT ist eine Script-Datei, die Befehle mit genau einem Argument mit Patternmatching ausstattet. Sie bildet das mit Hilfe des Befehls LIST nach.

```
SPAT:
.KEY COM/A,PAT/A,OPT1,OPT2,OPT3,OPT4
  ;Einlesen des Befehls, des Suchmusters und der Optionen für
  ;den DOS-Befehl. Der Befehl und das Suchmuster müssen
  ;eingegeben werden.
```

```
FAILAT 21
  ; » *" « meint ein Gänsefüßchen, weil verschachtelte
  ;Gänsefüßchen nicht erlaubt sind !
LIST >T:q<$$> <PAT> LFORMAT="<COM> *"%s%s*" <OPT1> <OPT2> <OPT3>
<OPT4>"
  ;LIST erzeugt eine temporäre Datei, in der eine
  ;Befehlsszeile pro gefundener zu bearbeitender Datei
  ;gespeichert wird (siehe auch LIST ... LFORMAT).
IF NOT FAIL
  EXECUTE T:q<$$>
ELSE
  ECHO "<pat> not found"
ENDIF
  ;Falls auf das Suchmuster passende Dateien im angegebenen
  ;Verzeichnis gefunden wurden, wird die erzeugte temporäre
  ;Scriptdatei ausgeführt,
  ;ansonsten wird eine Fehlermeldung ausgegeben.
FAILAT 10
```

DPAT

Format: DPAT <Befehl> <PATTERN> <ZIEL> [OPTIONEN]

Zweck: Patternmatching für Befehle mit zwei Argumenten.

Pfad: S:DPAT

Beschreibung:

DPAT unterstützt Befehle, die zwei Argumente verlangen, mit Patternmatching. Dazu benutzt DPAT die Mustererkennungsfähigkeit von LIST und erzeugt mit Hilfe der Ausgabeformatierung mit LFORMAT von LSIT eine temporäre Befehlsdatei, die mit EXECUTE ausgeführt wird.

```
DPAT:
.KEY COM/A,PAT/A,DIR/A,OPT1,OPT2,OPT3,OPT4
  ;Eingabe des Befehls, des Suchmusters, des Ziel-
  ;verzeichnisses und gegebenenfalls der Optionen für den
  ;Befehl
FAILAT 21
ECHO >ENV:qw<$$> "<DIR>" FIRST=256
  ;Setzen einer Hilfsvariablen im ENV:-Verzeichnis
  ;FIRST=256 ist ein fieser Trick,
  ;sauberer wäre hier: »ECHO >ENV:qw<$$> "<DIR>" LEN 1«
  ;/*** FALLUNTERSCHEIDUNG ***/
  ;wichtig für LFORMAT-Befehlszeichenkette für den Zugriffspfad
  ;des Zielverzeichnisses
  ; » *" « meint ein Gänsefüßchen, weil verschachtelte
```

```
  ;Gänsefüßchen nicht erlaubt sind !
IF $qw<$$> EQ ":"
  ;Verweis auf das Wurzelverzeichnis, des aktuellen
  ;Geräts
  LIST >T:q<$$> <PAT> LFORMAT="<COM> *"%s%s*" *"<DIR>%s*" <OPT1>
  <OPT2> <OPT3> <OPT4>"
  SKIP doit
ENDIF
IF $qw<$$> EQ "/"
  ;Verweis auf das übergeordnete Verzeichnis
  LIST >T:q<$$> <PAT> LFORMAT="<COM> *"%s%s*" *"<DIR>%s*" <OPT1>
  <OPT2> <OPT3> <OPT4>"
  SKIP doit
ENDIF
IF EXISTS <dir>
  ;Unterverzeichnis des aktuellen Verzeichnisses
  LIST >T:q<$$> <PAT> LFORMAT="<COM> *"%s%s*" *"<DIR>/%s*" <OPT1>
  <OPT2> <OPT3> <OPT4>"
ELSE
  ;Namen eines (logischen) Geräts
  LIST >T:q<$$> <PAT> LFORMAT="<COM> *"%s%s*" *"<DIR>*" <OPT1>
  <OPT2> <OPT3> <OPT4>"
ENDIF
LAB doit
IF NOT FAIL
  EXECUTE T:q<$$>
ELSE
  ECHO "<pat> not found"
ENDIF
  ;Falls auf das Suchmuster passende Dateien im angegebenen
  ;Verzeichnis gefunden wurden, wird die erzeugte temporäre
  ;Scriptdatei ausgeführt,
  ;ansonsten wird eine Fehlermeldung ausgegeben.
FAILAT 10
```

4.5 Push und Pop für das aktuelle Verzeichnis

Wer oft das aktuelle Verzeichnis zwischen zwei verschiedenen Verzeichnissen umschaltet, wird es bald leid sein, dauernd die kompletten Pfade einzugeben, wenn er zwischen den Verzeichnissen wechselt und wünscht sich eine Möglichkeit, das Verzeichnis, das er gerade verließ, auf einer Art Stapel zwischenzuspeichern.

PCD

Format: PCD [Verzeichnis]

Zweck: Zwischenspeichern des verlassenen aktuellen Verzeichnisses, um dahin zurückkehren zu können.

Pfad: S:PCD

Beschreibung:

PCD mit einem Verzeichnis speichert das verlassene Verzeichnis zwischen und macht das angegebene zum neuen aktuellen Verzeichnis. PCD ohne Argument lädt das letzte verlassene Verzeichnis und macht es zum aktuellen Verzeichnis.

```
PCD:
.KEY DIR
IF "<dir>" EQ ""
  ;wenn kein neues aktuelles Verzeichnis angegeben wurde altes
  ;zurückholen.
  ; » "" « meint hier eine leere Zeichenkette
  ASSIGN OFROM<$$>: ""
  ;aktuelles Verzeichnis abspeichern
  ; » "" « meint hier das alte aktuelle Verzeichnis !!!
  CD FROM<$$>:
  ;altes zurückholen
  ASSIGN FROM<$$>: OFROM<$$>:
  ASSIGN OFROM<$$>:
  ;temporäre Variablen tauschen
ELSE
  ;sonst neues Verzeichnis setzen
  ASSIGN FROM<$$>: ""
  ;aktuelles Verzeichnis speichern
  ; » "" « meint hier das alte aktuelle Verzeichnis !!!
  cd <dir>
  ;neues setzen
ENDIF
```

5 Die Mountlist

Die Mountlist ist eine Textdatei, in der Informationen über Geräte eingetragen werden, die in das System mit dem Befehl MOUNT eingebunden oder für das Dateisystem nutzbar gemacht werden sollen. Sie dient dem Betriebssystem sozusagen als Briefkasten oder Ablagebrett für Informationen, die der Benutzer bereitstellen muß, damit Amiga-DOS das Gerät, den Handler oder das Dateisystem benutzen kann. So ist es zum Beispiel bei nicht autobootfähigen Festplatten notwendig, die technischen Daten der Festplatte und das gewünschte Filing-System in die Mountlist einzutragen.

Das Betriebssystem sucht die Mountlist auf dem logischen Gerät DEVS:, das der Anwender entweder individuell korrekt mit ASSIGN setzen muß, oder vom System automatisch auf »SYS:DEVS« gesetzt wird. (Lesen Sie genaueres über die DOS-Befehle im Kapitel zwei und über Script-Dateien in Kapitel drei nach.)

Ein Mountlist-Eintrag für ein bestimmtes Gerät besteht aus einer Liste von Schlüsselwörtern, denen ein Wert zugeordnet wird. Ein Schlüsselwort kann auch entfallen, woraufhin diesem Schlüsselwort automatisch ein voreingestellter Wert vom Betriebssystem zugewiesen wird.

Ein Mountlist-Eintrag beginnt immer mit dem Namen des Gerätes, der mit einem Doppelpunkt abgeschlossen wird. Der Gerätename ist beliebig und kann vom Benutzer auch nach eigenem Geschmack geändert werden. Zum Beispiel wird anstatt »RAD:« für die autobootfähige RAM-Disk gerne der Name »VD0:« verwendet, der für englisch »virtual disk« (simulierte Platte) mit der Einheitnummer 0 steht. Auf ein Schlüsselwort folgt ein Gleichheitszeichen und auf dessen rechter Seite der Wert, der an das Schlüsselwort zugewiesen wird. Die einzelnen Einträge werden durch Semikolon getrennt oder jeweils in eine neue Zeile geschrieben. Kommentare können im Text eingefügt werden, und werden wie in der Programmiersprache C mit der Zeichenfolge »/*« eingeleitet und mit »*/« abgeschlossen. Geschachtelte Kommentare sind nicht erlaubt, die erste schließende »Kommentarklammer« beendet das Ausblenden des Kommentares. Ein Mountlist-Eintrag wird mit einem »Lattenzaun (#) beendet, der am Schluß des Eintrags in einer eigenen Zeile angefügt wird.
Folgende Schlüsselwörter werden ausgewertet:

Handler= Gibt den Handler an, der für das Gerät verwendet werden soll (Als Beispiel *Handler= L:Aux-Handler*)

File-System= Gibt an, welches Filing-System von einem Gerät benutzt werden soll (zum Beispiel File-System= L:Fast-File-System). Wird kein Eintrag angegeben, wird für ein Gerät zur Datenspeicherung das alte Filing-System benutzt.

Device= Gerätetreiber, der für das Gerät benutzt werden soll (zum Beispiel Device= scsidev.device)

Priority= Setzt die Priorität, unter der ein Dateisystem oder ein Handler ablaufen soll. Für Dateisysteme ist der Wert 10 gut geeignet, für Handler sollte die Priorität auf 5 gesetzt werden.

Unit= Gibt die hardwaremäßig eingestellte Nummer eines Geräts an. Die Angabe der Gerätenummer ist notwendig, wenn auf ein bestimmtes von mehreren Geräten zugegriffen werden soll.

Flags= Ist für den Funktionsaufruf »Open-Device« notwendig, der den Gerätetreiber lädt und in das System einbindet. Der Wert ist normalerweise 0 und für Diskettenlaufwerke 1.

Surfaces= Gibt die Anzahl der Oberflächen eines Gerätes an. Das ist für Diskettenlaufwerke und Festplatten wichtig, die die Schnittstellennorm ST 506 erfüllen, da hier die Benutzung der Plattenoberflächen vom Gerätetreiber verwaltet werden muß. SCSI-Festplatten benötigen diesen Eintrag nur für die optimale Performance, da sie die Oberflächen selbst verwalten.

BlocksPerTrack= Anzahl der Blöcke pro Spur (Auch hier nehmen es SCSI-Platten im Zweifel nicht so genau. Mit beiden Einträgen läßt sich aber auch einiges Tricksen, siehe unten).

Reserved= Die Anzahl der Blöcke für den Boot-Block muß mindestens 2 sein.

PreAlloc= Einige Festplatten speichern Daten für ihre Verwaltung in den letzten Blöcken auf sich selbst. Deshalb muß dem Filing-System mitgeteilt werden, diese Blöcke für die ausschließliche Nutzung durch die Platte selbst freizuhalten. Platten dieser Bauart sind recht selten. Ob und auf welchen Wert dieser Eintrag gesetzt werden muß, ist der Beschreibung der Festplatte zu entnehmen. Der Wert von Pre-Alloc ist auf 0 voreingestellt.

Interleave= Der Interleavefaktor gibt an, in welchem Abstand physikalisch aufeinanderfolgende Blöcke gelesen werden können. Optimal wäre es, wenn jede Festplatte alle Blöcke eines Sektors direkt nacheinander lesen könnte. Manche Platten jedoch besitzen diese Fähigkeit nicht. Bei diesen Modellen müssen nach einem gelesenen Block mindestens eine bestimmte Anzahl von Blöcken übersprungen werden, bevor wieder ein Block gelesen werden kann. Das heißt um zwei physikalisch aufeinanderfolgende Blöcke zu lesen, brauchen diese Festplatten zwischen beiden Lesezugriffen eine vollständige Umdrehung der Plattenober-

fläche. Dieser Wert ist von der jeweiligen Festplatte abhängig. Er muß in der Beschreibung der Festplatte angegeben sein.

LowCyl= Nummer des ersten physikalischen Zylinders, den eine Partition verwenden soll.

HighCyl= Nummer des letzten Zylinders, der von einer Festplatte verwendet werden soll. Er ist von der Bauart der Festplatte abhängig.

Stacksize= Größe des Stapelspeichers, der dem Handler- oder Filing-System-Prozeß zugeordnet werden soll.

Buffers= Anzahl der Zwischenspeicher, die nach Anmelden des Geräts verwendet werden sollen (siehe Kapitel zwei ADDBUFFERS).

BufMemType= Legt fest, welcher Speichertyp für die Zwischenspeicher des Geräts verwendet werden soll. 0 und 1 = egal, was vorhanden ist, wird benutzt; 2 und 3 = CHIP-RAM, muß verwendet werden; 4 und 5 = FAST-RAM, soll verwendet werden. Vorsicht, falls keines vorhanden!

Mount= Ist der angegebene Wert eine positive Zahl, so wird der Handler für das Gerät sofort geladen, ist die Zahl negativ wird der Handler erst beim ersten Zugriff auf das Gerät geladen.

MaxTransfer= Die maximale Größe in Bytes, der am Stück übertragenen Datenblöcke vom Speichermedium wird festgelegt. Manche Controller oder Festplatten benötigen das bei Verwendung des Fast-Filing-System, weil dies nahezu beliebig große Dateien am Stück übertragen kann, was einige Controller oder Festplatten nicht unterstützen. Der Maxtransfer hängt von der verwendeten Hardware ab und kann zum Glück meist entfallen, da er je nach Wert die Performance der Platte beschneidet.

Mask= Legt den Speicherbereich fest, der von DMA-Controllern für die Datenübertragung mit Direct-Memory-Access benutzt werden darf. So bekommt der A2090-Controller von Commodore Schwierigkeiten, wenn er zusammen mit einer Turbokarte (68020) von CSA verwendet wird, die die Adressierung des vollen Adreßbereichs von vier Gigabyte des 68020 32-Bit-Prozessors unterstützt. In diesem Fall muß der Eintrag Mask = 0xFFFFFF gesetzt werden, der das DMA auf die 24-Bit Adreßbreite des 68000 beschränkt.

GlobVec= Der BCPL-Kern des Amiga-DOS beschert viele Umstände und Ärger. Zum korrekten Start eines Prozesses, dessen Code vom Amiga-BCPL-Compiler erzeugt wurde, ist es notwendig, diesem Prozeß einen privaten Global-Vektor (Sprungzeiger) mitzugeben. Der Wert 0 bewirkt die Bereitstellung eines solchen Global-Vektor. Für Handler, die nicht in BCPL geschrieben wurden (Assembler oder C, vielleicht auch Modula?!?), ist kein spezieller Global-Vektor nötig, was für selbstgeschriebene Handler der Normalfall ist. Geben Sie dann einfach –1 an. Die Daseinsberechtigung dieses Schlüsselworts endet hoffent-

lich bald mit der Entfernung aller BCPL-Relikte in einer der zukünftigen Betriebssystemversionen, da für keinen Programmierer auf dem Amiga der BCPL-Compiler mehr erhältlich ist, und keine Programme mehr in BCPL entwickelt werden können (Gott sei Dank!).

Startup= Ein BPTR (BCPL-Zeiger), der auf einen BSTR (BCPL-Zeichenkette) zeigt, die beim Start an ein Gerät, einen Handler oder ein Dateisystem übergeben wird.

BootPri= Mit diesem Parameter kann der Benutzter festsetzen, welche Priorität ein Gerät bei einem Reboot haben soll, daß mit dem Befehl MOUNT im System beim ersten Start nach dem Einschalten des Amiga angemeldet wird. Ein Beispiel für ein solches Gerät ist die Recoverable-RAM-Disk RAD:. Der Wert darf zwischen –128 für kein Booten möglich und 127 liegen. Der Anwender sollte diesen Wert vorher sehr genau überdenken. Sollte das Gerät nämlich nicht mehr von dem Versuch abzubringen sein von der RAD: zu booten, obwohl das nicht beabsichtigt war und der Start von einer Diskette erfolgen sollte, hilft nur noch der Griff zum Netzschalter.

DosType= Gibt die Art des Dateisystems an, im Moment sind dafür zwei verschiedene Einträge möglich. Für das Fast-Filing-System wird der Wert 0x444F5301 (Hexadezimal für DOS1) verwendet. Für das alte Filing-System wird der Wert 0x444F5300 (DOS0) verwendet.

Um die Anwendung verschiedener Mountlist-Einträge genauer zu verstehen, sollen jetzt noch einige Beispiele diskutiert werden.

Auf der Original-Workbench 1.3 sind diese Einträge vorhanden:

```
/*  MountList for V1.3 */
/*  Im Eintrag für den neuen Console-Handler wird die Priorität des
    Handlerprozesses und die Größe des Stapelspeichers für diesen
    Prozeß festgesetzt.  */

NEWCON:
     Handler   = L:Newcon-Handler
     Priority  = 5
     StackSize = 1000
#

/*  Der Speak-Handler ist ein Beispiel für einen in C geschriebenen
    Handler, der nicht Teil des Dateisystems ist. Der Eintrag
    GlobVec= ▒1 legt fest, daß für den Handler kein privater Sprung-
    zeiger benötigt wird, da er nicht in BCPL geschrieben ist.  */
```

```
SPEAK:
     Handler   = L:Speak-Handler
     Stacksize = 6000
     Priority  = 5
     GlobVec   = -1
#
/*  Der Aux-Handler benutzt eine ungepufferte Datenübertragung über
    die serielle Schnittstelle. Mit ihm kann zum Beispiel eine SHELL
    auf ein an der seriellen Schnittstelle angeschlossenes Terminal
    umgeleitet werden.  */

AUX:
     Handler   = L:Aux-Handler
     Stacksize = 1000
     Priority  = 5
#

/*  Obwohl der Pipe-Handler Datenströme von der Standardausgabe
    eines SHELL-Prozesses als Standardeingabe eines zweiten SHELL-
    Prozesses umleitet, ist er kein Bestandteil des Dateisystems.
    Das Gerät PIPE: ist sehr nützlich, wenn Ausgaben auf die SHELL
    als Eingabe eines zweiten Prozesses verwendet werden sollen,
    ohne daß temporäre Dateien angelegt werden. Ein zweiter Vorteil
    ist, daß beide Programme im Multitasking gleichzeitig gestartet
    werden können, und ihre Ausführung von der PIPE synchronisiert
    wird. Der schreibende Prozeß wird angehalten, wenn der Puffer
    voll ist, der lesende wenn er leer ist.  */

PIPE:
     Handler   = L:Pipe-Handler
     Stacksize = 6000
     Priority  = 5
     GlobVec   = -1
#

/*  Dies ist ein Beispieleintrag für eine RAM-Disk, die ihren Inhalt
    bei einem Reboot nicht verliert und von der das System einen
    Reboot ausführen kann (siehe Kapitel drei die Startup-Sequence).
    Um eine gesamte Diskette mit Diskcopy auf die RAD: kopieren zu
    können, ist es wichtig für Surfaces, BlocksPerTrack, LowCyl und
    HighCyl Werte einzutragen, die den Daten eines Diskettenlauf-
    werks gleichen. BufMemType= 1 gibt an, daß CHIP- oder FAST-RAM
    vom System für die Zwischenspeicher beliebig gewählt werden
    dürfen, wobei hier die Priorität auf der Nutzung des FAST-RAM
    liegt. Der Wert für die Anzahl der Puffer ist bewußt niedrig
    gewählt, da das Kopieren von Daten von der RAM-Disk in einen im
```

```
    RAM befindlichen Puffer viel eher noch eine Geschwindigkeitsein-
    buße bedeutet, da die Daten im Speicher hin- und herkopiert
    werden. Die RAM-Disk hat naturgemäß eine wesentlich höhere
    Geschwindigkeit als eine physikalisch vorhandene Disketten-
    station oder Festplatte, so daß auf ihre Daten nahezu direkt
    zugegriffen werden kann.  */

RAD:
     Device         = ramdrive.device
     Unit           = 0
     Flags          = 0
     Surfaces       = 2
     BlocksPerTrack = 11
     Reserved       = 2
     Interleave     = 0
     LowCyl         = 0  ;  HighCyl = 79
     Buffers        = 5
     BufMemType     = 1
#

/*  Für eine optimale Performance sollte die RAD: das Fast-Filing-
    System benutzen. Dabei ist zu beachten, daß ihre Bootpriorität
    auf -128 gesetzt wird (kein Autoboot erlaubt), da Amiga-DOS zum
    jetzigen Zeitpunkt mit der RAD: noch nicht vom Fast-Filing-
    System booten kann. */

RAD:
     Device         = ramdrive.device
     Unit           = 1
     Flags          = 0
     Surfaces       = 2
     BlocksPerTrack = 11
     Reserved       = 2
     Interleave     = 0
     LowCyl         = 0  ;  HighCyl = 79
     Buffers        = 5
     BufMemType     = 1
     GlobVec        = -1                 /* Required for FFS  */
     StackSize      = 4000               /* Required for FFS  */
     DosType        = 0x444F5301         /* Required for FFS  */
     FileSystem     = L:fastfilesystem   /* Required for FFS  */
     BootPri        = -128               /* Autoboot disabled */
#

/*  Dieser Eintrag dient zur Anmeldung mit dem Befehl MOUNT eines
    extern angeschlossenen Diskettenlaufwerks als DF2:.  */
```

```
DF2:
     Device          = trackdisk.device
     Unit            = 2
     Flags           = 1
     Surfaces        = 2
     BlocksPerTrack  = 11
     Reserved        = 2
     Interleave      = 0
     LowCyl          = 0  ;  HighCyl = 39
     Buffers         = 20
     BufMemType      = 3
#
/*  FAST: ist ein Beispieleintrag für eine Festplatte, die am A2090A
    betrieben werden könnte. Die Partition DH0: wurde mit PREP
    erzeugt, alle weiteren können durch den entsprechenden Eintrag
    hier in der Mountlist definiert, und wie DH0: nach dem ersten
    Formatieren benutzt werden. Die Partition DH0: wird mit dem
    normalen Filing-System benutzt, damit Autoboot von der Fest-
    platte möglich wird. Manche Festplattentreiber von Fremdherstel-
    lern benötigen mehr Stapelspeicher.  */
FFS1:
     Device          = hddisk.device
     File-System     = 1:Fast-File-System
     Unit            = 1
     Flags           = 0
     Surfaces        = 4
     BlocksPerTrack  = 17
     Reserved        = 2
     Interleave      = 0
     LowCyl          = 21  ;  HighCyl = 800
     Buffers         = 30
     GlobVec         = -1
     BufMemType      = 1
     Mount           = 1
     DosType         = 0x444F5301
     StackSize       = 4000
#
/*  Festplatten können natürlich noch ganz anders vergewaltigt
    werden. Es ist nämlich auch denkbar, auf einer Festplatte ein
    Diskettenlaufwerk zu simulieren: */
```

```
FLOP:
     Device         = hddisk.device
     Unit           = 1
     Flags          = 0
     Surfaces       = 2
     BlocksPerTrack = 11
     Reserved       = 2
     Interleave     = 0
     LowCyl         = 800  ;  HighCyl = 879
     Buffers        = 30
     GlobVec        = -1
     BufMemType     = 3
     Mount          = 1
     DosType        = 0x444F5300
     StackSize      = 4000
#

/*  Für eine SCSI-Festplatte von Quantum mit einem Controller von
    Great Valley Products würde ein Eintrag so aussehen:  */

DH0:
 Device         = scsidev.device
 /* spezieller Treiber, der mit dem Kontroller ausgeliefert wird */
 Unit           = 1            /* Platte mit SCSI ID = 0; 80.1 MB */
 Flags          = 0
 Reserved       = 2
 Buffers        = 32
 BufmemType     = 1
 Surfaces       = 2
 BlocksPerTrack = 32
 LowCyl         = 1 ; HighCyl = 2560
 GlobVec        = -1           /* Required for FFS */
 StackSize      = 4000         /* Required for FFS */
 MaxTransfer    = 8192
 /* Wird in der nächsten Version des Treibers entfernt */
 DosType = 0x444F5301          /* Required for FFS */
 File-System = L:fastfilesystem /* Required for FFS */
#

/*  Der Amiga 2000 erkennt extern angeschlossene Diskettenlaufwerke
    erst ab der Nummer DF2:. Wenn der Amiga 2000 ein internes und
    ein externes Laufwerk besitzt, der Benutzer das externe Laufwerk
    aber gerne als DF1: ansprechen würde, so kann er folgenden Ein-
    trag verwenden, der das Diskettenlaufwerk mit der physikalischen
    Nummer zwei unter der (logischen) Nummer eins ansprechbar
    macht.  */
```

```
DF1:
     Device          = trackdisk.device
     Unit            = 2
     Flags           = 1
     Surfaces        = 2
     BlocksPerTrack  = 11
     Reserved        = 2
     Interleave      = 0
     LowCyl          = 0  ;  HighCyl = 79
     Buffers         = 20
     BufMemType      = 3
#
```

6 Der Makro-Assembler

Dieses Kapitel beschreibt den Amiga-DOS-Makro-Assembler (von MetaComCo), und ist für Benutzer des Asm68K, der Public Domain ist und zum Beispiel auf einer Fish Disk erhältlich ist, gleichermaßen nützlich. Zusätzlich gibt es einen kurzen Überblick über den Mikroprozessor MC 68000 und die Struktur von Objekt-Files. Dieses Kapitel ist für Programmierer geschrieben, die sich bereits mit Assemblerprogrammierung auf einem anderen Computer beschäftigt haben. (Bei terminologischen Unklarheiten sei auf das folgende Kapitel »Der Linker« verwiesen)

6.1 Einführung zum Mikroprozessor MC 68000

Dieser Abschnitt beschreibt den Mikroprozessor 68000. Er soll Ihnen helfen, die später vorgestellten Konzepte zu verstehen. Er setzt voraus, daß Sie bereits Erfahrung im Umgang mit anderen Prozessoren haben. Der für den Prozessor verfügbare Speicher besteht aus zwei Teilen:

```
den internen Registern auf dem Chip selbst und
dem Hauptspeicher außerhalb des Chips.
```

Von den 17 Registern des 68000 stehen ständig 16 zur Verfügung. Acht von ihnen sind sogenannte Datenregister, die mit D0 bis D7 bezeichnet werden. Die anderen acht Register werden Adreßregister genannt und mit A0 bis A7 bezeichnet. Jedes Register hat eine Breite von 32 Bit. In vielen Fällen können Sie beliebige Register verwenden, manche Operationen fordern aber explizit einen bestimmten Typ. Zum Beispiel kann jedes Register für interne Operationen mit den Datentypen Wort (16 Bit) oder Langwort (32 Bit) oder zur indizierten Adressierung des Hauptspeichers verwendet werden. Dagegen wird zur Operation mit dem Byte-Datentyp (8 Bit) ein Datenregister verlangt; zur direkten Adressierung des Hauptspeichers ein Adreßregister als Stack-Pointer oder Basis-Pointer. Das Register A7 ist der Stack-Pointer. Dabei handelt es sich in Wirklichkeit um zwei Register: im Supervisor-Modus ist es der System-Stack-Pointer, im Anwender-Modus der Benutzer-Stack-Pointer für die Anwendungen. Der Hauptspeicher besteht aus einer großen Anzahl einzelner Bytes. Jedes Byte hat eine Identifizierungs-Nummer, seine Adresse. Der 68000 kann über 16 Millionen Byte direkt adressieren. Dabei können Operationen mit einem Byte, Wort oder Langwort durchgeführt werden. Ein Wort besteht aus zwei zusammengehörigen Bytes. Dabei hat das erste Byte immer eine gerade Adresse. Ein Langwort besteht aus vier Bytes. Auch dabei hat das erste Byte eine gerade Adresse. Die Adresse des Langwortes ist die

Adresse des niedersten geraden Bytes. Neben den einzelnen Daten, die der Rechner manipulieren soll, enthält der Hauptspeicher auch die Befehle zu den Manipulationen, die er ausführen soll. Jeder Befehl belegt ein bis fünf Worte im Speicher, eines für den eigentlichen Befehl und bis zu vier Operanden. Der Befehl enthält auch die Anzahl an zugehörigen Operanden. Die Operanden enthalten die Adresse der Speicherstellen, deren Werte verarbeitet werden sollen, und die Adressen, wohin das Ergebnis gespeichert werden soll. Der Prozessor führt einen Befehl im Speicher nach dem anderen aus, so wie Sie ein Musikstück Note für Note vom Blatt abspielen. Ein spezielles Register, der Programm-Counter (PC), enthält die Adresse des als nächsten auszuführenden Befehls. Einige Befehle, Sprünge oder Verzweigungen genannt, werfen diese Ordnung durcheinander. Mit ihnen wird der Rechner angewiesen, als nächstes den Befehl an einer anderen angegebenen Adresse auszuführen. Somit kann der Rechner etwas wiederholt ausführen oder den Fortgang des Programmes von einem Wert abhängig machen. Informationen über das Ergebnis von Operationen (z.B. Übertrag, Überlauf, bei Überschreitung des Zahlenbereiches etc.), und den Status des Prozessors (z.B. Tracebit-Modus, Supervisor-Modus) befinden sich im Status-Register (SR).

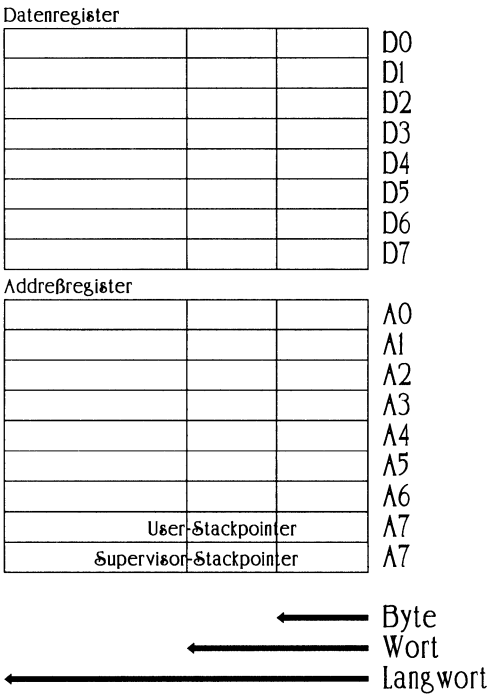


Bild 6.1: Die 68000-Prozessor-Register

6.2 Der Aufruf des Assemblers

Das Befehls-Muster für ASSEM lautet:

```
ASSEM "PROG=FROM/A,-O/K,-V/K,-L/K,-H/K,-C/K,-I/K"
```

alternativ kann das Muster so beschrieben werden:

```
ASSEM <ursprungfile> [-o <objektfile>][ -I <listingfile>]
      [-v <ausgabefile>][-h <headerfile>][-c <Optionen>]
      [-i <include Dir.liste>]
```

Der Assembler erzeugt kein Listingfile oder Objektfile, wenn Sie es nicht anfordern. Alle Meldungen, die der Assembler während seiner Arbeit erzeugt, werden auf dem Bildschirm ausgegeben, wenn Sie kein Ausgabefile benannt haben. Um ein File an den Beginn Ihres Ursprungscodes einzubinden, wird die Option –h <filename> verwandt. Den gleichen Effekt hat der Befehl INCLUDE "<filename>" in der ersten Zeile Ihres Quellcodes. Um die Liste der Directories zu spezifizieren, die vom Assembler auf der Suche nach einem Include-File durchsucht werden sollen, wird die Option »–i« verwendet. Die einzelnen Directories werden in dem Befehl durch ein Leerzeichen, ein Komma oder das (+) getrennt. Verwenden Sie ein Leerzeichen zur Trennung, muß die gesamte Liste der Directories in Anführungszeichen (") stehen. Die Directories werden in der genannten Reihenfolge nach dem File durchsucht. Das File, das vom Assembler eingebunden werden soll, muß im aktuellen oder in einem der mit »–i« bezeichneten Directories stehen. Soll zum Beispiel das File fred.asm assembliert und dabei das File intern/incl, das File include/asm und das File extern/incl eingebunden werden, lauten die drei möglichen Befehle:

```
assem fred.asm -i intern/incl, include/asm, extern/incl
assem fred.asm -i intern/incl+include/asm+extern/incl
assem fred.asm -i "intern/incl include/asm extern/incl"
```

Mit der Option »–c« lassen sich noch einige weitere Optionen angeben. Jede besteht aus einem Buchstaben in Groß- oder Kleinschrift, zum Teil gefolgt von einer Zahl. Hier nun die Beschreibung der Optionen:

```
S   erzeugt einen Symbol-Dump als Teil des Objektfiles.
D   unterdrückt die Ausgabe lokaler Labels in den Symbol-Dump.
    (In C beginnt jedes lokale Label mit einem ».«.)
C   ignoriert Groß- oder Kleinschreibung bei Labels.
X   erzeugt am Ende des Listingfiles eine Cross-Referenz-Tabelle.
```

Beispiele:

```
ASSEM fred.asm -o fred.o
```

assembliert das File fred.asm und erzeugt das Objektfile fred.o.

```
ASSEM fred.asm -o fred.o -l fred.erst
```

assembliert das File »fred.asm«, erzeugt das Objektfile »fred.o« und das Listingfile »fred.erst«.

6.3 Aufbau des Quellcodes

Ein File, das vom Assembler bearbeitet werden soll, besteht aus einer Serie von Zeilen, die folgendes enthalten können:

- Kommentare oder Leerzeilen
- Ausführbare Befehle
- Direktiven an den Assembler

6.3.1 Kommentare

Kommentare können auf drei verschiedenen Wegen in ein Programm eingebracht werden:

1. Geben Sie irgendwo in der Zeile einen Strichpunkt (;) ein. Alles nach diesem Strichpunkt ist Kommentar. Das folgende Beispiel zeigt einen solchen Kommentar:

   ```
   CMPA.L A1, A2 ; Sind die Pointer gleich?
   ```

2. Geben Sie einen Stern (*) als erstes Zeichen einer Zeile ein. Die gesamte Zeile enthält nur Kommentar. Das folgende Beispiel zeigt einen solchen Kommentar:

   ```
   * Die ganze Zeile enthält Kommentar
   ```

3. Geben Sie nach einem vollständigen Befehl mindestens ein Leerzeichen ein und dann den Kommentar:

   ```
   MOVEQ #10,D0 schreibt den Ausgangswert in Register D0
   ```

Alle Leerzeilen werden vom Assembler als Kommentare aufgefaßt.

6.3.2 Ausführbare Instruktionen

Ausführbare Instruktionen der Quelldatei haben generell dieses Format:

```
[<label>] <opcode> [<operand>[,<operand>]]...[<kommentar>]
```

Um ein Feld von einem anderen zu trennen, können Sie ein Leerzeichen oder die [TAB]-Taste verwenden. Beide erzeugen ein Trennzeichen. Mehrere Trennzeichen hintereinander sind erlaubt.

6.3.2.1 Das Label-Feld

Ein Label ist ein vom Programmierer definiertes Symbol, das:

a) in der ersten Spalte der Zeile beginnt und vom Opcode durch mindestens ein Leerzeichen getrennt ist oder

b) irgendwo in einer Zeile steht, dann aber von einem Doppelpunkt (:) beendet wird.

Geben Sie ein Label an, muß es das erste Zeichen einer Zeile sein, das kein Leerzeichen ist. Der Assembler setzt dann den PC (Programmzähler), das ist das Register, das die Adresse des nächsten Befehls im Speicher enthält, auf die Zeile mit dem Label. Labels können allen Befehlen und nahezu allen Direktiven an den Assembler zugewiesen werden, oder sie können auch allein in einer Zeile stehen. In Abschnitt 6.7 finden Sie weitere Informationen zu Labels bei Direktiven. Jedes Label darf nur einmal vergeben werden. Der Name eines Labels darf keine Makro-Namen, keine Device-Namen, keine Befehle oder Register-Bezeichnungen enthalten.

6.3.2.2 Lokale Labels

Die lokalen Variablen wurden zusätzlich zu den normalen Labels von Motorola eingeführt. Sie werden in der Form »nnn$« angegeben und sind nur zwischen zwei normalen Labels gültig. Hier ein Beispiel:

```
Label    Opcode    Operanden

FOO:     MOVE.L    D6,D0
1$:      MOVE.B    (A0)+,(A1)+
         DBRA      D0,1$
         MOVEQ     #20,D0
BAA:     TRAP      #4
```

In diesem Fall ist das Label »1$« nur erreichbar, wenn es nach der Zeile mit dem Label FOO und vor der Zeile mit dem Label BAA verwendet wird. Der Name »1$« kann in diesem Fall zwischen zwei anderen Labels jederzeit wieder verwendet werden.

6.3.2.3 Das Opcode-Feld

Das Feld Opcode kommt nach dem Label und ist von diesem durch mindestens ein Leerzeichen getrennt. In diesem Feld können stehen:

a) Opcodes für den MC 68000, wie im MC6800 User Manual definiert
b) Direktiven an den Assembler
c) Makro-Aufrufe

Zusätzlich können den Opcodes, die mehr als eine Datengröße verarbeiten können, noch Kennungen für die jeweilige Datengröße übergeben werden. Diese werden vom Operator nur durch einen Punkt (.) getrennt. Diese Kennungen sind:

B	Größe ist ein Byte (8 Bit)
W	Größe ist Wort (16 Bit)
L	Größe ist Langwort (32 Bit) oder Kennung für langen Sprung
S	Kennung für kurzen Sprung

Die angegebene Größe muß natürlich zum jeweiligen Befehl passen.

6.3.2.4 Das Operanden-Feld

Falls vorhanden, enthält dieses Feld einen oder mehrere Operanden zu dem Befehl oder der Direktive. Der Operand wird durch mindestens ein Leerzeichen vom Opcode getrennt. Sind mehrere Operanden angegeben, werden diese durch Kommata getrennt. Das Operanden-Feld endet mit einem Leerzeichen oder mit dem Zeichen für neue Zeile, das erscheint, wenn [Return] betätigt wird. Leerzeichen zwischen den Operanden müssen in diesem Fall nicht eingegeben werden.

6.3.2.5 Das Kommentar-Feld

Dieses Feld wurde bereits in Abschnitt 6.3.1 besprochen.

6.4 Ausdrücke

Ein Ausdruck ist eine Kombination von Symbolen, Konstanten, algebraischen Operatoren und Klammern, mit denen Sie Befehle oder Direktiven für den Assembler erstellen.

6.4.1 Operatoren

An Operatoren stehen zur Verfügung (aufgelistet nach der Wertigkeit):

1. Minus als Vorzeichen und logisches NOT (- und ~)
2. LinksSHIFT und RechtsSHIFT (<< und >>)
3. Logisches UND und logisches ODER (& und !)
4. Multiplikation und Division (* und /)
5. Addition und Subtraktion (+ und -)

Um die Wertigkeit eines Operators zu umgehen, wird der ganze Term in Klammern eingeschlossen. Operationen gleicher Wertigkeit werden von links nach rechts ausgeführt. Bitte beachten Sie, daß ein Leerzeichen als Trennung zwischen den Feldern steht.

6.4.2 Operand/Operator-Kombinationen

In der folgenden Tabelle wird die Zulässigkeit eines Operators zu den Operanden dargestellt. Die Tabelle zeigt alle möglichen Operand/Operator-Kombinationen auf. Dabei steht »A« für einen absoluten Operanden und »R« für einen relativen. Ein »x« zeigt einen Fehler an und »op« steht für Operator. Das Vorzeichen Minus und die logischen Operationen sind nur für absolute Operanden zulässig.

Operatoren	Operanden			
	A op A	R op R	A op R	R op A
+	A	x	R	R
-	A	A	A	R
*	A	x	x	x
/	A	x	x	x
&	A	x	x	x
|	A	x	x	x
>>	A	x	x	x
<<	A	x	x	x

Tabelle 6.1: Zulässige Operand/Operator-Kombinationen

6.4.3 Symbole

Ein Symbol ist ein String aus bis zu 30 Zeichen. Das erste Zeichen eines Symboles muß eines der folgenden sein:

- Ein alphanumerisches Zeichen (A-Z oder a-z)
- Ein Unterstreichungsstrich (_)
- Ein Punkt (.)

Die restlichen Zeichen können außerdem noch aus Ziffern (0–9) bestehen. Groß- oder Kleinschrift wird beachtet! Das Symbol »fred« ist nicht gleich dem Symbol »FRED« oder »FRed«! Dies gilt jedoch nicht, wenn Sie beim Aufruf des Assemblers die Option -C angegeben haben. Alle Opcodes, Befehle, Argumente oder Direktiven können in Groß- oder Kleinbuchstaben eingegeben werden. Ein Symbol kann aus 30 Zeichen bestehen, alle werden beachtet. Geben Sie mehr als 30 Zeichen ein, werden die überzähligen abgeschnitten und eine Fehlermeldung ausgegeben. Ein einem Register-Namen mit EQUR gleichgestelltes Label wird ebenso in Groß- oder Kleinschrift erkannt. Die Namen von Befehlen, Direktiven, Devices, Registern und die Sonderzeichen CCR, SR, SP und USP dürfen Sie nicht als Symbole verwenden. Drei Typen von Symbolen sind möglich:

Absolut
- a) Das Symbol wurde mit SET oder EQU einem absoluten Wert gleichgestellt.

Relativ
- a) Das Symbol wurde mit SET oder EQU einem relativen Wert gleichgestellt.
- b) Das Symbol wurde als Label benutzt.

Register
- a) Das Symbol wurde durch EQUR als Register-Name gesetzt.

Das besondere Symbol »*« hat den Wert und Typ des aktuellen Programm-Zählers, also der Adresse des gerade ausgeführten Befehls oder der Direktive.

6.4.4 Zahlen

Ziffern werden als Teil eines Ausdrucks oder als einzelner Wert angegeben. Zahlen haben immer absolute Werte und entsprechen einem der folgenden Formate:

```
Dezimal:
                (eine Reihe dezimaler Ziffern)
                Beispiel: 12345

Hexadezimal:
                ($ gefolgt von hexadezimalen Zeichen)
                Beispiel: $89AC2

Oktal:
                (@ gefolgt von oktalen Ziffern)
                Beispiel: @73413

Binär:
                (% gefolgt von Einsen und Nullen)
                Beispiel: %01101101

ASCII-Literale:
                (Bis zu vier ASCII-Zeichen in »'« eingeschlossen)
                Beispiel: «ABCD« «*«
```

ASCII-Strings mit weniger als vier Zeichen werden rechtsbündig verwendet und mit Null auf vier Zeichen ergänzt. Der Apostroph innerhalb eines Strings wird durch einen zweiten kenntlich gemacht. Ein Beispiel ist: 'Holger' 's'

6.5 Adressierungs-Arten

Die genaue Beschreibung der unterschiedlichen Adressierungs-Arten des MC68000 finden Sie in jedem guten Handbuch zu diesem Prozessor. Die folgende Tabelle gibt einen Überblick:

Adressierung	Beschreibung und Beispiele	
Dn	Daten-Register	direkte Adressierung
	Beispiel:	MOVE D0,D1
An	Adreß-Register	direkte Adressierung
	Beispiel:	MOVEA A0,A1
(An)	Adreß-Register	indirekte Adressierung
	Beispiel:	MOVE D0,(A1)
(An)+	Adreß-Register	indirekte Adressierung mit Postincrement

Adressierung	Beschreibung und Beispiele	
	Beispiel:	MOVE (A7)+,D0
-(An)	Adreß-Register	indirekte Adressierung mit Predekrement
	Beispiel:	MOVE D0,-(A7)
a(An)	Adreß-Register	indirekt Adressierung, verschoben
	Beispiel:	MOVE 20(A0),D1
a(An,Xn)	Adreß-Register	indirekte Adressierung, indiziert
	Beispiele:	MOVE 0(A0,D0),D1
		MOVE 120(A0,D6.W),D4
a	absolut, kurz (16 Bit)	
	Beispiel:	MOVE $1000,D0
a	absolut, lang (32 Bit)	
	Beispiel:	MOVE $10000,D0
r	Programm-Zähler (PC) relativ mit Verschiebung	
	Beispiel:	MOVE ABC,D0 (ABC ist relativ)
r(Xn)	PC relativ, indiziert	
	Beispiel:	MOVE ABC(D0.L),D1 (ABC ist relativ)
#a	Datenliteral	
	Beispiel:	MOVE #1234,D0
USP	Spezieller Adreß-Modus	
CCR	Spezieller Adreß-Modus	
SR	Spezieller Adreß-Modus	
	Beispiele:	MOVE A0,UPS
		MOVE D0,CCR
		MOVE D1,SR

Tabelle 6.2: Adressierungs-Arten

Adressierungen beziehen sich immer auf ein bestimmtes Byte. Befehle und Operationen mit der Größe Wort und Langwort benötigen jedoch mehr als ein Byte. Diese Daten müssen bei einer geraden Adresse beginnen. In der obigen Tabelle steht »Dn« für ein Daten-Register D0–D7, ein »An« für ein Adreß-Register A0–A7, SP und PC, ein »a« für einen absoluten Wert, ein »r« für einen relativen Wert und »Xn« steht für An oder Dn mit den Optionen ».W« oder ».L«.

6.6 Verschiedene Befehlstypen

Einige Befehle (zum Beispiel ADD, CMP) besitzen spezielle Varianten. Adreßvarianten beziehen sich auf Adreßregister als Ziel. Immediate- und Quick-Varianten arbeiten mit literalen Daten. Memory-Varianten benötigen Postinkrement-Adressen für beide Operanden.

Um eine dieser Varianten einzusetzen, werden dem Mnemonic die Zusätze »A«dreß, »Q«uick, »I«mmediate oder »M«emory angehängt. Dann benutzt der Assembler die jeweilige Sonderform des Befehls, wenn vorhanden. Ansonsten gibt er eine Fehlermeldung aus.

Haben Sie keine der Varianten definiert, verwendet der Assembler automatisch diese Formen, wenn das möglich ist. Das gilt nicht für die Variante »Q«. Der Assembler wandelt zum Beispiel die Form

```
ADD.L A2,A1
```

automatisch in die folgende kompaktere Form um:

```
ADDA.L A2,A1
```

6.7 Direktiven

Alle Assembler-Direktiven (mit Ausnahme von DC und DCB) sind Befehle an den Assembler, sie werden nicht direkt in den Objektcode umgesetzt. Zu Beginn dieses Abschnittes finden Sie eine Tabelle mit allen Direktiven (Tabelle 6.3), geordnet nach Funktionen; später werden alle Direktiven genauer beschrieben. Diese Beschreibungen sind ebenfalls nach Funktionen geordnet.

Beachten Sie bitte, daß nicht allen Direktiven ein Label zugeordnet werden darf. Zum Beispiel ist zu EQU ein Label erlaubt, möglich ist es bei RORG, nicht zulässig aber bei LLEN oder TTL. Die Zulässigkeit eines Labels wird in der genauen Beschreibung vermerkt.

Direktive	Beschreibung
Assembler-Kontrolle	
SECTION	Programm-Abschnitt
RORG	Verschiebbarer Anfang
OFFSET	Definiert Offsets
END	Ende des Programmes
Symbol-Definition	
EQU	Weist konstanten Wert zu
EQUR	Weist Register konstanten Wert zu
REG	Weist konstanten Wert zu
SET	Weist variablen Wert zu
Daten-Definition	
DC	Definiert Konstante
DCB	Definiert konstanten Block
DS	Definiert Speicherplatz-Reservierung

Direktive	Beschreibung
Listing-Kontrolle	
PAGE	Seitenvorschübe im Listing ein
LIST	Listing einschalten
NOLIST(NOL)	Listing ausschalten
SPC n	fügt n Leerzeilen ein
NOPAGE	Seitenvorschub im Listing: aus
LLEN n	Setzt die Anzahl der Zeichen pro Zeile (60<=n<=132)
PLEN n	Setzt die Anzahl der Zeilen pro Seite (24<=n<=100)
TTL	Setzt Programm-Name (max. 40 Zeichen)
NOOBJ	Verhindert Ausgabe von Objektcode
FAIL	Erzeugt Assembler-Fehler
FORMAT	Keine Funktion
NOFORMAT	Keine Funktion
Bedingte Assemblierung	
CNOP	Bedingtes NOP zur Angleichung
IFEQ	Assembliert, wenn Ausdruck 0
IFNE	Assembliert, wenn Ausdruck nicht 0
IFGT	Assembliert, wenn Ausdruck >0
IFGE	Assembliert, wenn Ausdruck >=0
IFLT	Assembliert, wenn Ausdruck <0
IFLE	Assembliert, wenn Ausdruck <=0
IFC	Assembliert, wenn Strings gleich
IFNC	Assembliert, wenn Strings ungleich
IFD	Assembliert, wenn Symbol definiert
IFND	Assembliert, wenn Symbol nicht definiert
ENDC	Ende der bedingten Assemblierung
Direktiven zu Makros	
MAKRO	Definiert Makro-Name
NARG	Spezielles Symbol
ENDM	Ende des Makro
MEXIT	Verläßt Makro vorzeitig
Externe Symbole	
XDEF	Definiert externes Symbol
XREF	Nimmt Bezug auf externes Symbol
Allgemeine Direktiven	
INCLUDE	Bindet File in Quellcode
MASK2	Keine Funktion
IDNT B	Benennt Programm-Einheit

Tabelle 6.3: Assembler-Direktiven

6.7.1 Direktiven zur Assembler-Kontrolle

SECTION

Muster: [<label>] SECTION <name>[,<typ>]

Beschreibung:

Diese Direktive weist den Assembler an, den Zähler zum letzten Befehl der benannten Sektion zurückzusetzen. (Der Zähler wird Null, wenn er zum ersten Mal benutzt wird.)

<name> ist ein String, eventuell in »'« eingeschlossen.

<typ> ist, wenn angegeben, eines der folgenden Schlüsselworte:

CODE (der Default) zeigt an, daß die Sektion verschiebbaren Code enthält.

DATA zeigt an, daß die Sektion nur fest definierte Daten enthält.

BSS zeigt an, daß die Sektion variable Daten enthält.

Der Assembler kann bis zu 255 Sektionen verwalten. Der Assembler beginnt mit einer nicht benannten Sektion vom Typ CODE. Der Assembler weist das optionale Argument <label> nach der Ausführung von SECTION dem PC zu. Ist eine Sektion unbenannt, wird das Schlüsselwort CODE zum Namen dieser Sektion.

RORG

Muster: [<label>] RORG <absolutwert>

Beschreibung:

Die Direktive RORG ändert den Wert des PC zu <absolutwert> Bytes hinter den Start der aktuellen verschiebbaren Sektion. Der Assembler weist den folgenden Befehlen verschiebbare Adressen zu. Um Adressierungen in verschiebbaren Sektionen vornehmen zu können, muß die Adressierung PC relativ mit Verschiebung angewandt werden. Die Wertzuweisung an <label> ist ansonsten gleich der bei SECTION.

OFFSET

Muster: OFFSET <absolutwert>

Beschreibung:

Um eine Tabelle von Offset-Werten mit der Direktive DS anzulegen, die bei <absolutwert> beginnt, benötigen Sie die OFFSET-Direktive. In einer OFFSET-Tabelle definierte Symbole werden intern verwaltet, dürfen jedoch keine Befehle enthalten, die Opcodes erzeugen. Um eine OFFSET-Sektion zu beenden, können RORG, OFFSET, SECTION oder END verwendet werden.

END

Muster: [<label>] END

Beschreibung:

Die END-Direktive teilt dem Assembler das Ende des Quellcodes mit. Alle weiteren Befehle oder Direktiven werden ignoriert. Erreicht der Assembler im ersten Durchlauf ein END, beginnt er den zweiten. Wird vor dem END die End-of-File-Marke erreicht, wird eine Fehlermeldung ausgegeben. Geben Sie ein Label ein, wird dem PC der Wert des Labels zugewiesen, bevor END ausgeführt wird.

6.7.2 Direktiven zur Symbol-Definition

EQU

Muster: <label> EQU <ausdruck>

Beschreibung:

Der Wert des Operanden <ausdruck> wird dem Label <label> zugewiesen. Die Zuweisung ist absolut, Sie dürfen dieses Label innerhalb des Programmes kein zweites Mal definieren.

Sie sollten in <ausdruck> nicht auf andere, noch folgende Definitionen verweisen. Auf die mit EQU zugewiesenen Symbole darf vor der Definition nicht zugegriffen werden.

EQUR

Muster: <label> EQUR <register>

Beschreibung:

Mit dieser Direktive wird ein Symbol <label> einem Register des Prozessors gleichgestellt. Diese Direktive ist nur für Adreß- und Daten-Register zulässig. Die Definition ist absolut, das Label darf kein zweites Mal definiert werden. Der Assembler unterscheidet bei dem Label nicht zwischen Groß- und Kleinschreibung.

Auf die mit EQUR zugewiesenen Symbole darf vor der Definition nicht zugegriffen werden.

REG

Muster: <label>REG<registerliste>

Beschreibung:

Der Assembler weist Label eine Liste von Registern nach dem mit MOVEM verwandten Muster zu. Dabei ist <registerliste> von folgender Form:

```
R1 [-R2][/R3[-R4]]...
```

SET

Muster: <label> SET <ausdruck>

Beschreibung:

Mit SET wird dem <label> der Wert von <ausdruck> zugewiesen. SET unterscheidet sich von EQU dadurch, daß die Zuweisung des Wertes zum Symbol nicht dauerhaft ist. Sie kann später im Programm jederzeit verändert werden. Sie sollten in <ausdruck> nicht auf andere noch folgende Definitionen verweisen. Auf die mit SET zugewiesenen Symbole darf vor der Definition nicht zugegriffen werden.

6.7.3 Direktiven zur Daten-Definition

DC

Muster: [<label>] DC[.<größe>] <liste>

Beschreibung:

Mit DC wird eine Liste von Konstanten in den Speicher plaziert. Mehrere Konstanten können, durch Kommata getrennt, angegeben werden. Die Werte müssen dabei jedoch in der mit <größe> angegebenen Anzahl an Bits speicherbar sein. Wird <größe> nicht angegeben, ist ».W« voreingestellt.

Geben Sie als <größe> ».B« an, steht zur Speicherung auch die Option ASCII-String zur Verfügung. Das ist eine beliebig lange Folge von ASCII-Zeichen, eingefaßt in Apostrophe »'«. Soll der Apostroph innerhalb des Strings verwendet werden, müssen zwei hintereinander eingegeben werden. Ist als <größe> ».W« oder ».L« angegeben, füllt der Assembler den Wert bis zur vollen Größe auf.

DCB

Muster: [<label>] DC[.<größe>] <absolutwert>,<ausdruck>

Beschreibung:

Mit DCB wird eine Anzahl <absolutwert> von Bytes, Worten oder Langworten (abhängig von <größe>) in den Speicher plaziert. Jedes dieser Bytes, Worte oder Langworte erhält denselben Inhalt, nämlich den Wert von <ausdruck>. DCB<.größe> n, <ausdruck> entspricht n Wiederholungen der Direktive DC<größe> <ausdruck>.

DS

Muster: [<label>] DS[.<größe>] <absolutwert>

Beschreibung:

DS reserviert <absolutwert> Bytes, Worte oder Langworte (abhängig von <größe>). Wird <größe> nicht angegeben, ist ».W« eingestellt. DS initialisiert diesen Speicher jedoch nicht. Die folgende Direktive ergibt zum Beispiel einen reservierten Speicherbereich von 1024 Byte.

```
DS.L 256
```

6.7.4 Listing-Kontrolle

PAGE

Muster: PAGE

Beschreibung:

PAGE beginnt eine neue Seite im Assembler-Listing. PAGE selbst erscheint nicht im Ausgabe-Listing.

LIST

Muster: LIST

Beschreibung:

Die Direktive LIST weist den Assembler an, ein formatiertes Listing der folgenden Instruktionen in das Listing-File zu schreiben. Der Befehl wird ausgeführt, bis der Assembler auf ein NOLIST oder END trifft. Diese Direktive ist nur wirksam, wenn beim Aufruf des Assemblers ein Listing-File angegeben wurde. Die Direktive LIST selbst erscheint nicht im Ausgabe-Listing.

NOLIST oder NOL

Muster: NOLIST

Beschreibung:

NOL oder NOLIST hebt die Wirkung von LIST auf – die folgenden Instruktionen erscheinen nicht im Ausgabe-Listing. Auch diese Direktive selbst erscheint nicht im Ausgabe-Listing.

SPC

Muster: SPC <zahl>

Beschreibung:

SPC erzeugt <zahl> Leerzeilen im Assembler-Listing. Die Direktive SPC selbst erscheint nicht im Ausgabe-Listing.

NOPAGE

Muster: NOPAGE

Beschreibung:

Mit NOPAGE wird die Ausgabe von Seitenvorschüben und von Kopfzeilen im Assembler-Listing unterdrückt.

LLEN

Muster: LLEN <zahl>

Beschreibung:

Mit LLEN wird die Länge einer Zeile im Assembler-Listing-File eingestellt. Der Wert <zahl> muß zwischen 60 und 132 liegen, er kann nur einmal eingestellt werden. Grundeinstellung ist 132 Zeichen/Zeile. Die Direktive LLEN erscheint nicht im Ausgabe-Listing.

PLEN

Muster: PLEN <zahl>

Beschreibung:

Mit PLEN wird die Anzahl der Zeilen pro Seite des Assembler-Listings eingestellt. <zahl> muß zwischen 24 und 100 liegen. Der Wert kann nur einmal verändert werden. Grundeinstellung ist 60 Zeilen pro Seite.

TTL

Muster: TTL <namestring>

Beschreibung:

TTL gibt dem assemblierten Programm einen Namen, der in der Kopfzeile des Assembler-Listings ausgedruckt wird. Der Name beginnt mit dem ersten Zeichen nach TTL, das kein Leerzeichen ist, und darf nicht länger als 40 Zeichen sein. Die Direktive TTL erscheint nicht im Ausgabe-Listing.

NOOBJ

Muster: NOOBJ

Beschreibung:

Mit NOOBJ wird die Ausgabe des Objektcodes in ein File verhindert, auch wenn Sie beim Aufruf des Assemblers ein Objektcode-File angegeben haben.

FAIL

Muster: FAIL

Beschreibung:

Fail weist den Assembler an, einen Fehler in dieser Eingabezeile anzuzeigen.

FORMAT

Muster: FORMAT

Beschreibung:

Der Assembler akzeptiert diese Direktive, führt jedoch keine Aktion aus. FORMAT wurde implementiert, um die Kompatibilität mit anderen Assemblern zu gewährleisten.

NOFORMAT

Muster: NOFORMAT

Beschreibung:

Der Assembler akzeptiert diese Direktive, führt jedoch keine Aktion aus. NOFORMAT wurde implementiert, um die Kompatibilität mit anderen Assemblern zu gewährleisten.

6.7.5 Bedingte Assemblierung

CNOP

Muster: [<label>] CNOP <zahl>,<zahl>

Beschreibung:

Diese Direktive gehört nicht zum Standard-Befehlssatz eines MC68000-Assemblers. Sie erlaubt es, einen Bereich des erzeugten Codes an einer Grenze auszurichten. Jede Datenstruktur und jeder Einsprungpunkt kann so an einer Langwortgrenze ausgerichtet werden.

Der erste Ausdruck repräsentiert einen Versatz, der zweite die gewünschte Ausrichtung. Der Code wird mit dem angegebenen Versatz bei der nächsterreichbaren Grenze ausgerichtet. Die folgende Anweisung richtet also den Code am nächsten Langwort aus (setzt den PC also an die nächste durch vier teilbare Adresse):

```
CNOP 0,4
```

Die nächste Anweisung hingegen richtet den erzeugten Code an der Adresse 2 Byte hinter der nächsten Langwortgrenze aus:

```
CNOP 2,4
```

IF

Muster: IFxx <absolutwert>

Beschreibung:

Mit IFxx wird die bedingte Assemblierung gesteuert. Alle nach dieser Direktive folgenden Befehle werden nur assembliert, wenn die Bedingung erfüllt ist. Der Bereich der bedingten Assemblierung geht bis zur Direktive ENDC. Dahinter fährt die Assemblierung auf alle Fälle wieder fort. Bedingungen können auch geschachtelt werden. Dabei muß aber beachtet werden, daß die jeweils zuletzt aufgestellte Bedingung zuerst mit einem ENDC beendet werden muß. Jede Bedingung muß mit einem eigenen ENDC versehen sein.

Für »xx« (die eigentliche Bedingung) können Sie verwenden:

EQ assembliert, wenn Ausdruck = 0NE assembliert, wenn Ausdruck <> 0GT assembliert, wenn Ausdruck >0GE assembliert, wenn Ausdruck >=0LT assembliert, wenn Ausdruck <0LE Assembliert, wenn Ausdruck <=0

IFC und IFNC

Muster: IFC <string>,<string>

IFNC <string>,<string>

Beschreibung:

IFC assembliert die folgenden Anweisungen (bis zum ENDC) nur, wenn <string1> und <string2> gleich sind, IFNC assembliert, wenn sie nicht gleich sind. Dabei ist jeder <string> eine Kette von ASCII-Zeichen, der von Apostrophen begrenzt wird. Ist die Bedingung NICHT erfüllt, wird der Teil des Quellcodes übersprungen, der zwischen dem IF(N)C und der Direktive ENDC steht.

IFD und IFND

Muster: IFD <symbolname>

IFND <symbolname>

Beschreibung:

Der Bereich bis zum nächsten ENDC wird assembliert, wenn das Symbol <symbolname> definiert (IFD) oder nicht definiert (IFND) ist.

ENDC

Muster: ENDC

Beschreibung:

Um den Bereich der bedingten Assemblierung abzuschließen, wird ENDC eingesetzt. Damit werden alle acht oben beschriebenen Direktiven zur bedingten Assemblierung (die mit IF... beginnen) beendet.

6.7.6 Direktiven zu Makros

MAKRO

Muster: <label> MAKRO

Beschreibung:

Mit MAKRO beginnt die Definition eines Makros. ENDM beendet eine Makro-Definition. Ein Label muß angegeben werden, das der Assembler als Makro-Namen verwendet. Beim Aufruf des Labels als Operand wird statt des Labels der Quellcode zwischen dem entsprechenden MAKRO und ENDM eingesetzt. Ein Makro kann aus Opcodes, Assembler-

Direktiven oder weiteren Makros bestehen. Ein (+) im Listing markiert den Code, der von einem Makro erstellt wurde. Jedem Makro-Aufruf können Argumente übergeben werden. Sie werden, durch Kommata getrennt, hinter dem Makro-Namen eingegeben. Enthält ein Argument ein Leerzeichen (zum Beispiel einen String), muß das ganze Argument zwischen die Zeichen Kleiner als (<) und Größer als (>) gesetzt werden.

Der Assembler speichert den Quellcode, den Sie in das Makro schreiben (nach der Direktive MAKRO und vor der Direktive ENDM), als Makro. Das Makro kann jeden normalen Quellcode enthalten. Zusätzlich hat das Symbol »\« (Backslash) Bedeutung. Ein <Backslash> gefolgt von einer Zahl n versteht der Assembler als Aufforderung, das n-te Argument an dieser Stelle in den Code einzubauen. Wurde das n-te Argument beim Aufruf des Makros nicht angegeben, wird nichts eingesetzt. Ein <Backslash> gefolgt vom Zeichen <@> weist den Assembler an, den Text ».nnn« zu erstellen, wobei nnn die Anzahl der Aufrufe einer »\@«-Kombination ist. Dies wird normalerweise eingesetzt, um gleiche Labels innerhalb eines Makros zu erzeugen. Makro-Definitionen dürfen nicht verschachtelt werden. Jedoch kann in einem Makro jederzeit ein vorher definiertes Makro aufgerufen werden. Es dürfen allerdings nur zehn Makro-Ebenen auf diese Weise ineinander eingebaut werden.

Der Einbau eines Makros wird beendet, wenn das Ende des Makros erreicht, oder die Direktive MEXIT gefunden wird. Ein Makro unterscheidet sich von einem Unterprogramm einer höheren Programmiersprache deutlicher, als es im ersten Moment den Eindruck erweckt. Ein Unterprogramm wird angesprungen, ausgeführt, und dann wird die Programmausführung an dem Punkt des Hauptprogrammes weitergeführt, der nach dem Unterprogramm-Aufruf kommt. Ein Makro dagegen ist nur eine gute Kopierfunktion. Der gesamte Quellcode des Makros wird bei jedem Aufruf gleichsam an anderer Stelle in den Quellcode hineinkopiert. (Zeitraubende) Sprünge innerhalb des Programmes sind dazu nicht nötig.

NARG

Muster: NARG

Beschreibung:

NARG wird nur in Makros verwendet. NARG entspricht immer der Nummer des letzten Argumentes, das beim Aufruf eines Makros angegeben wurde. Außerhalb eines Makros hat NARG den Wert Null.

ENDM

Muster: ENDM

Beschreibung:

Beendet die Definition eines Makros.

MEXIT

Muster: MEXIT

Beschreibung: Mit der Kombination MEXIT (und bedingter Assemblierung) kann ein Makro vorzeitig – das heißt vor Erreichen der Direktive ENDM – abgebrochen werden.

6.7.7 Externe Symbole

XDEF

Muster: XDEF <label>[,<label>...]

Beschreibung:

XDEF folgen ein oder mehrere absolute oder verschiebbare Labels. Jedes hier genannte Label erzeugt eine externe Symbol-Definition. Auf diese Symbole kann sich ein anderes Modul (das zum Beispiel in einer höheren Programmiersprache geschrieben wurde) beziehen. Die entsprechende Verbindung stellt der Linker her. Verwenden Sie diese Direktive oder auch XREF, kann der vom Assembler erzeugte Objektcode nicht ohne weiteres Linken ausgeführt werden.

XREF

Muster: XREF <label>[,<label>...]

Beschreibung:

Ein oder mehrere Labels müssen hinter XREF eingegeben werden. Diese Labels dürfen an anderer Stelle im Programm nicht definiert werden. Diese Labels bezeichnen Adressen in anderen Modulen, die erst später vom Linker berechnet und festgelegt werden.

Externe Symbole werden normalerweise folgendermaßen eingesetzt: Um eine Routine innerhalb eines Programm-Segmentes als externe Routine zu definieren, wird ein Label an ihren Anfang gesetzt. Dieses Label wird dann der Direktive XDEF nach außen gereicht. Ein anderes Programm ruft diese Routine auf, indem es mit XREF ein Label definiert und das so deklarierte Label anspringen kann, als stände es im selben Programm.

6.7.8 Allgemeine Direktiven

INCLUDE

Muster: INCLUDE "<filename>"

Beschreibung:

Die Direktive INCLUDE erlaubt das Einbinden externer Files in den Quellcode. Der Filename wird als Operand übergeben. Der Assembler verwendet den Inhalt dieses Files dann, als würde er statt der INCLUDE-Direktive in dem gerade assemblierten File stehen. INCLUDE kann bis zu drei Ebenen tief verschachtelt werden, wenn die Filenamen, wie gezeigt, in Anführungszeichen gesetzt werden.

INCLUDE wird besonders dann benutzt, wenn Makros oder EQU-Zuweisungen in gleicher Form in mehreren Programmen eingebaut werden sollen. Diese Definitionen werden dazu zweckmäßigerweise in ein File geschrieben, das dann in alle Programme mit INCLUDE eingebunden wird. Zu Beginn dieses Files können Sie die Direktive NOLIST, zum Ende LIST einsetzen, um die Ausgabe dieser sich stets wiederholenden Zeilen in das Listfile zu verhindern.

Das einzubindende File wird zuerst im aktuellen Directory, dann in der beim Aufruf des Assemblers übergebenen »–I Dir.liste« gesucht.

MASK2

Muster: MASK2

Beschreibung:

Der Assembler akzeptiert diese Direktive, führt jedoch keine Aktion aus.

IDNT

Muster: IDNT <string>

Beschreibung:

Eine Programm-Einheit, die aus einer oder mehreren Sektionen besteht, muß einen Namen haben. Dieser wird mit der Direktive IDNT übergeben. Findet der Assembler diese Direktive nicht, bekommt das Programm einen Null-String als Namen.

6.7.9 Die Objekt-File-Struktur

Als Objekt-File wird die Ausgabe eines Assemblers oder eines Compilers bezeichnet. Um das Objekt-File zum ausführbaren Programm zu machen, werden zunächst alle externen Referenzen aufgelöst. Diese Arbeit übernimmt der Linker. Ein Objekt-File besteht aus einer oder mehreren Programmeinheiten. Jede Programmeinheit beginnt mit einem Header. Dann folgt eine Anzahl Module, die wiederum aus Blöcken unterschiedlichen Typs bestehen. Jeder Block beginnt mit einem Langwort, das den Typ des Blocks enthält. Dann folgt eine Null oder weitere Langworte. Jeder Block wird auf volle Langwortlänge erweitert. Der Header einer Programmeinheit beginnt ebenfalls mit einem Block dieses Formates.

Eine Programmeinheit enthält also:

- den Header-Block der Programmeinheit
- eine Reihe von Modulen

Die Grundstruktur eines Moduls besteht aus:

- einem Speicherblock für den Modul-Namen
- einem relozierbaren Block
- einem Block mit Informationen zur Relokation

- einem Block mit den Informationen zu externen Symbolen
- einem Block mit der Symboltabelle
- einem Block mit Debugging-Informationen
- dem Endblock

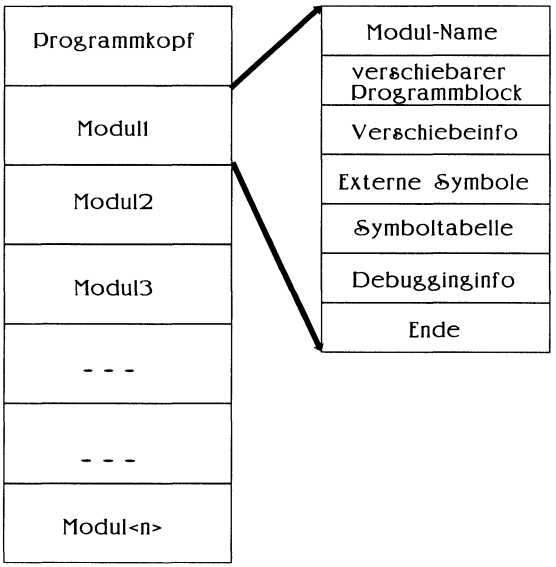


Bild 6.2: Objekt-File-Struktur

Alle Blöcke mit Ausnahme des Endblocks können allerdings auch entfallen.

Die folgenden Abschnitte beschreiben das Format jedes dieser Blöcke. Die Werte, die als Angabe des Blocktyps übergeben werden, sind in dezimaler und hexadezimaler Form nach dem Modul-Namen abgedruckt. Zum Beispiel wird dem _hunk_unitr (hunk bedeutet Modul) als Definition der Wert 999 dezimal oder 3E7 hexadezimal zugewiesen.

6.7.10 hunk_unit (999/3E7)

Dieses Modul zeigt den Beginn einer Programmeinheit an. Das erste Wort enthält den Typ des Moduls, gefolgt von der Länge des Namens der Programmeinheit in Langworten, wiederum gefolgt vom Namen der Programmeinheit in Langworten, bei Bedarf bis zur nächsten Langwortgrenze aufgefüllt mit Nullen. Grafisch dargestellt sieht hunk_unit wie folgt aus:

```
hunk_unit
N
N
Langworte mit dem Namen
s...
```

Bild 6.3: hunk_unit (999/3E7)

6.7.11 hunk_name (1000/3E8)

Dieser Block definiert den Namen des Moduls. Namen sind optional. Findet der Linker mehrere Module mit gleichem Namen, werden sie zu einem Modul verknüpft. Beachten Sie bitte, daß 8- oder 16-Bit-externe-Referenzen nur unter Modulen mit gleichem Namen aufgelöst werden können. Jede externe Referenz in einem ladbaren File, die auf ein anderes Modul verweist, muß als 32-Bit-Referenz übergeben werden. Die Module eines ladbaren Files werden nicht hintereinander, sondern eventuell verteilt in den Speicher geladen. Sie können deshalb auch nicht davon ausgehen, daß sie untereinander nur 32 Kbyte Abstand haben. Bedenken Sie bitte auch, daß alle Blöcke auf eine ganze Zahl von Langworten aufgefüllt werden. Grafisch dargestellt sieht hunk_name so aus:

```
hunk_name
N
N
Langworte mit dem Namen
...
```

Bild 6.4: hunk_name (1000/3E8)

6.7.12 hunk_code (1001/3E9)

Ein solcher Hunk definiert einen Block, der Code enthält und in den Speicher geladen und eventuell reloziert werden kann. Sein Format sieht wie folgt aus:

```
hunk_code
N
N
Langworte mit dem Code
...
```

Bild 6.5: hunk_code (1001/3E9)

6.7.13 hunk_data (1002/3EA)

Definiert einen initialisierten Datenblock, der in den Speicher geladen und dort reloziert werden kann. Der Linker verändert diese Blöcke nicht, wenn sie Teil eines Overlay-Knotens sind. Während des Overlay-Vorgangs werden sie also erneut in den Speicher geladen. Grafisch dargestellt sieht hunk_data wie folgt aus:

```
hunk_data
N
N
Langworte mit Daten
...
```

Bild 6.6: hunk_data (1002/3EA)

6.7.14 hunk_bss (1003/3EB)

Mit diesem Hunk wird ein nicht initialisierter Block im Arbeitsspeicher definiert, der später nachgeladen wird. Diese Blockform wird zum Beispiel für Stacks oder für FORTRAN-COMMON-Blöcke verwendet. Innerhalb dieser Blöcke kann nicht reloziert werden. Symboldefinitionen sind aber zulässig. Grafisch dargestellt sieht hunk_bss wie folgt aus:

```
hunk_bss
N
```

Bild 6.7: hunk_bss (1003/3EB)

Dabei ist N die Blocklänge in Langworten. Beim Laden wird der Speicher von BSS-Blöcken mit Null beschrieben. Der relozierbare Block innerhalb eines Moduls muß vom Typ hunk_code, hunk_data oder hunk_bss sein.

6.7.15 hunk_reloc32 (1004/3EC)

Ein hunk_reloc32-Block liefert Informationen zur 32-Bit-Relokation, die der Linker mit dem jeweils aktuellen relozierbaren Block durchführt. Die Informationen zur Relokation beziehen sich auf eine Stelle im aktuellen Modul oder auf eine innerhalb derselben Programmeinheit. Alle Module innerhalb einer Programmeinheit sind – beginnend mit Null – durchnumeriert. Der Linker addiert die Basis-Adresse des angegebenen Moduls zu den Langworten im vorhergehenden relozierbaren Block, der eine Liste mit Offsets enthält. Diese Offset-Liste enthält nur bestimmte Module und als Abschluß eine Null. Grafisch dargestellt sieht hunk_reloc32 wie folgt aus:

```
hunk_reloc32
N1
Modul Nummer 1
N1
Offsets
...
N2
Modul Nummer 2
N2
Offsets
...
```

```
Nn
Modul Nummer n
NnOffsets
...
0
```

Bild 6.8: hunk_reloc32 (1004/3EC)

6.7.16 hunk_reloc16 (1005/3ED)

Ein hunk_reloc16-Block liefert Informationen zur 16-Bit-Relokation, die der Linker mit dem jeweils aktuellen relozierbaren Block durchführt. Diese Information bezieht sich auf 16-Bit-PC-relative Referenzen auf andere Module innerhalb der Programmeinheit. Referenzen dürfen sich nur auf gleichnamige Module beziehen, da der Linker nur diese zu einer Einheit verkettet, die zusammen geladen werden. Das Format von hunk_reloc16 entspricht dem von hunk_reloc32.

6.7.17 hunk_reloc8 (1006/3EE)

Ein hunk_reloc8-Block liefert Informationen zur 8-Bit-Relokation, die der Linker mit dem jeweils aktuellen relozierbaren Block durchführt. Diese Information bezieht sich auf 8-Bit-PC-relative Referenzen auf andere Module innerhalb der Programmeinheit. Referenzen dürfen sich nur auf gleichnamige Module beziehen, da der Linker nur diese zu einer Einheit verkettet, die zusammen geladen werden. Das Format von hunk_reloc8 entspricht dem von hunk_reloc32.

6.7.18 hunk_ext (1007/3EF)

Dieser Block enthält Informationen zu externen Symbolen. Eingetragen werden sowohl die definierten Symbole als auch die externen Symbole, die dieses Modul aus anderen benötigt. Hier das Format:

```
hunk_ext
Symbol-Dateneinheit
Symbol-Dateneinheit
...
0
```

Bild 6.9: hunk_ext (1007/3EF)

Dabei wird eine Symbol-Dateneinheit für jedes Symbol verwandt. Der Block endet mit einem Langwort mit dem Wert Null. Jede Symbol-Dateneinheit enthält ein Byte zur Typdefinition, die Länge des Symbol-Namens in drei Bytes, den Symbol-Namen selbst und weitere Daten. Die Länge des Symbol-Namens geben Sie in Langworten an, der Symbol-Name selbst wird mit Nullen auf das nächste Langwort aufgefüllt.

Das Typ-Byte definiert das Symbol als Definition oder Referenz. Amiga-DOS verwendet für Symbol-Definitionen die Werte 0–127, die Werte 128–255 für Referenzen.

Folgende Werte sind definiert:

Name	Wert	Bedeutung
ext_symb	0	Symbol-Tabelle - siehe Symbol-Block weiter unten
ext_def 1		Relozierbare Definition
ext_abs 2		Absolute Definition
ext_res 3		Residente Library-Definition
ext_ref32	129	32-Bit-Referenz zu einem Symbol
ext_common	130	32-Bit-Referenz zu einem COMMON-Block
ext_ref16	131	16-Bit-Referenz zu einem Symbol
ext_ref8	132	8-Bit-Referenz zu einem Symbol

Tabelle 6.4: Externe Symbole

Der Linker gibt bei anderen Werten eine Fehlermeldung aus. Bei ext_def wird nur ein Datenwort verwandt. Es enthält nur den Offset des Symbols relativ zum Beginn des Moduls. Ext_abs enthält ebenfalls nur ein Datenwort, den absoluten Wert, der zum Code hinzugefügt wird. Der Linker behandelt den Wert von ext_res wie den von ext_def, geht aber davon aus, daß der Modul-Name ein Library-Name ist, und kopiert den Namen in das ladbare File. Den Typ-Bytes ext_ref32, ext_ref16 und ext_ref8 folgen jeweils ein Wert und eine Liste von Referenzen, die als Offset vom Beginn des Moduls angegeben sind.

Der Typ ext_common hat die gleiche Struktur wie ein COMMON-Block. Der Linker verarbeitet als COMMON definierte Symbole wie folgt: Findet er eine Definition für ein Symbol, das eine Referenz zu COMMON hat, verwendet er diesen Wert. Andernfalls belegt er den für jede COMMON-Symbol-Referenz als maximal angegebenen bss-Speicherplatz.

Der Linker bearbeitet externe Referenzen abhängig vom Typ der dazugehörigen Definition. Absolute Werte werden zu Langworten oder Byte-Feldern addiert. Paßt dieser (vorzeichenbehaftete) Wert nicht zur Definition, wird ein Fehler gemeldet. Bei relozierbaren 32-Bit-Referenzen wird der Wert des zugehörigen Symbols addiert und ein Relocation-Record für den Lader erstellt. 16- und 8-Bit-Referenzen werden als PC-relativ behandelt und sind nur für Relokationen innerhalb gleichnamiger Module zulässig. Solche PC-relative Referenzen werden nicht aufgelöst, wenn die Referenz und die Definition zu weit voneinander entfernt sind. Der Linker kann auf residente Libraries nur zugreifen, wenn 32-Bit-Referenzen dafür definiert wurden.

Die Symbol-Dateneinheiten sehen wie folgt aus:

```
ext_def/abs/res
                Typ | Name Länge NL
                NL Langworte des Symbolnamens
                ...
                Symbolwert

ext_ref32/16/8
                Typ | Name Länge NL
                NL Langworte des Symbolnamens
                ...
                Anzahl der Referenzen NR
                NL Langworte der Symbolreferenzen
                ...

ext_common
                130 | Name Länge NL
                NL Langworte des Symbolnamens
                Größe des COMMON-Blocks
                Anzahl der Referenzen NR
                NR Langworte der Symbolreferenzen
```

Bild 6.10: Symbol-Dateneinheit

6.7.19 hunk_symbol (1008/3E0)

Damit Sie einen symbolischen Debugger verwenden können, können Sie mit diesem Typ von Block eine Symboltabelle an ein Modul anhängen. Werden mehrere Module gleichen Namens zusammengelinkt, verknüpft der Linker auch deren Symboltabellen. Symboltabellen werden nicht in den Speicher geladen. Bei Bedarf muß sie der Debugger von der Diskette lesen. Das Format des Symboltabellen-Blocks entspricht dem des externen Symbol-Informations-Blocks mit Symboltabellen-Einheiten für jeden verwendeten Namen. In den Symbol-Dateneinheiten wird der Typencode Null verwendet. Der Wert des Symbols ist der Offset des Symbols vom Anfang des Moduls. Grafisch dargestellt sieht hunk_symbol wie folgt aus:

```
hunk_symbol
Symbol-Dateneinheit
...
0
```

Bild 6.11: hunk_symbol (1008/3F0)

dabei hat jede Symbol-Dateneinheit folgendes Format:

```
0 | Name Länge NL
NL Langworte des Symbolnamens
...
Symbolwert
```

Bild 6.12: Symbol-Dateneinheit

6.7.20 hunk_debug (1009/3F1)

Amiga-DOS stellt diesen Debug-Block zur Verfügung, um weitere Informationen für Debugging-Zwecke in einem ladbaren File ablegen zu können. Compiler können zum Beispiel eine Beschreibung der Datenstrukturen zur Verfügung stellen, die von Hochsprachen-Debuggern benötigt werden. Diese Informationen können im Debug-Block enthalten sein. Amiga-DOS schreibt für diesen Block nur den Anfang vor. Er beginnt mit der Typ-Kennung hunk_debug als Langwort, im zweiten Langwort steht die Länge des Blocks. Das Format der restlichen Daten ist nicht definiert.
Grafisch dargestellt sieht hunk_debug wie folgt aus:

```
hunk_debug
N
N
Langworte mit Debugging-Informationen
...
```

Bild 6.13: hunk_debug (1009/3F1)

6.7.21 hunk_end (1010/3F2)

Dieser Block definiert das Ende des Moduls. Er enthält nur das Langwort hunk_end.

7 Der Linker ALINK

Dieses Kapitel beschreibt ALINK, den Linker des Amiga-DOS. Der Linker bindet ein oder mehrere Eingabe-Files zu einem (als Programm ausführbaren) Binär-File. Der Linker ist auch fähig, Programme, die mit dem Overlay-Prinzip (siehe Abschnitt 7.3) arbeiten, zu erzeugen. Außerdem werden hier einige Fachwörter und der Aufbau eines binären Load-Files, das vom Linker erstellt und vom Loader in den Hauptspeicher gelesen wird, erklärt. Getrennt von der Beschreibung der Load-Files finden Sie in diesem Kapitel Anmerkungen über den Einsatz gemeinsamer Symbole (common symbols), absoluter externer Referenzen und Programm-Einheiten.

7.0 Terminologie

Einige der in den folgenden Abschnitten verwendeten Fachbegriffe werden nun kurz erklärt.

Symbol

Vergleiche 6.4.3

Externe Referenzen

Sie können Namen verwenden, um Bezüge (Referenzen) zwischen einzelnen Programmeinheiten zu kennzeichnen. Unter »externen Referenzen« verstehen wir dabei Namen, die in einem Programmteil definiert werden, die aber aus anderen Teilen ebenfalls verwendet werden können. Da jeder einzelne Programmteil separat assembliert werden kann, können diesen externen Labels natürlich noch keine Adressen zugeordnet werden. Dies geschieht erst durch den Linker, der die einzelnen Programmteile (die durch den Assembler erzeugten Objekt-Files) zusammenlinkt.
Namen, die als externe Referenzen benutzt werden, können von der Datenstruktur her 16 Mbyte lang sein, werden jedoch vom Linker auf 255 Zeichen gekürzt. Linken Sie Objektfiles zu einem einzelnen ladbaren File, muß jeder einzelnen externen Referenz eine externe Symboldefinition zugeordnet sein. Diese externe Referenz kann vom Datentyp Byte, Wort oder Langwort sein. Externe Definitionen beziehen sich auf relozierbare (verschiebbare) Werte, absolute Werte oder residente Bibliotheken. Relozierbare Werte beziehen sich auf den PC-relativen Adreßmodus und werden vom Linker verwaltet. Wann immer ein

Programm relozierbare Referenzen mit der Datenlänge Langwort enthält, wird die Relokation erst beim Laden des Programmes durch einen speziellen Relokator des Amiga-Betriebssystems durchgeführt.

Objekt-File

Ein Assembler oder Compiler erzeugt ein binäres Ausgabe-File, Objekt-File genannt. Ein Objekt-File besteht aus einer oder mehreren Programmeinheiten und kann externe Referenzen zu anderen Modulen enthalten.

Ladbares File (Load-File)

Der Linker erzeugt ein binäres Ausgabe-File aus einer beliebigen Anzahl von Objekt-Files. Dieses Ausgabe-File wird auch »ladbares File« genannt. Ein ladbares File enthält keine unaufgelösten externen Referenzen mehr.

Programmeinheit (Program Unit)

Eine Programmeinheit ist das kleinste vom Linker bearbeitbare Element. Eine Programmeinheit kann ein oder mehrere Module (Hunks) enthalten; ein Objekt-File kann eine oder mehrere Programmeinheiten enthalten. Findet der Linker eine externe Referenz, während er die Bibliotheks-Funktionen durchforstet, wird die gesamte Programmeinheit, in der die Referenz definiert wird, in das ladbare File eingebunden. Ein Assembler erzeugt ein einziges Programm aus einem Quell-File (das ein oder mehrere Module enthalten kann). Ein Compiler erzeugt meist eine Programmeinheit für jedes Unterprogramm, Hauptprogramm oder jeden Datenblock. Die Numerierung der Module beginnt in jeder Programmeinheit mit Null, auf andere Programmeinheiten kann nur mit »externen Referenzen« verwiesen werden.

Module (Hunks)

Ein Modul besteht aus Daten- oder Programmblöcken, Relokations-Informationen und eventuell einer Liste von definierten oder benötigten externen Symbolen. Daten-Module enthalten initialisierte oder nicht initialisierte Daten (BSS). BSS-Module können externe Definitionen enthalten. Sie können jedoch keine Referenzen auf externe Symbole oder relozierbare Werte enthalten. Initialisierte Datenblöcke innerhalb eines Overlay-Files (siehe 8.3) sollten nicht geändert werden (d.h. wie Konstanten behandelt werden), da sie während des Overlay-Prozesses immer wieder neu von der Diskette gelesen werden können. Eventuelle Änderungen gehen dabei verloren.
Module können benannt werden oder unbenannt bleiben, und sie können eine Symbol-Tabelle für Debugging-Zwecke enthalten. Auch Informationen zum Debugging mit höheren Programmiersprachen können vom Linker eingebunden werden. Die Module innerhalb einer Programmeinheit bekommen alle eine Nummer. Das erste Modul trägt die Nummer Null.

Residente Libraries (immer verfügbare Bibliotheken)

Ladbare Files werden oft auch »libraries« (Bibliotheken) genannt. Solche Bibliotheken können entweder immer im Speicher bereitgehalten (residente Libraries) oder als Teil der Exec-Funktion OpenLibrary() bei Bedarf geladen werden. Auf residente Libraries können Sie in Ihrem Programm durch externe Referenzen verweisen. Diese Verweise werden in ein Modul geschrieben, das nur eine Liste der residenten Libraries ohne weiteren Code enthält. Dieses Modul erstellen Sie, indem Sie ein File assemblieren, das nichts außer absoluten externen Symbolen enthält, und dieses mit einem speziellen Programm bearbeiten, das die absoluten Definitionen in Definitionen für residente Libraries umwandelt. Der Linker verwendet den Modul-Namen dann als Namen der residenten Library und bindet entsprechende Aufrufe in den Programmcode ein, so daß bei der Programmausführung die residente Library automatisch geöffnet wird, bevor sie das erste Mal verwendet wird.

Scanned Libraries

Eine Scanned Library besteht aus Objekt-Files, die Programmeinheiten enthalten, die nur dann geladen werden, wenn durch eine externe Referenz auf sie verwiesen wird. Scanned Libraries sind also nicht resident, sondern werden erst dann geladen, wenn sie benötigt werden. Objekt-Files können als Libraries definiert und dem Linker als primäres Input-File übergeben werden. In diesem Fall werden alle Programmeinheiten, die die Bibliothek enthält, beim Linken mitverarbeitet. Beachten Sie, daß Objekt-Files verkettet werden können.

Knoten (Node)

Ein Knoten besteht aus mindestens einem Modul. Ein Overlay-Load-File enthält einen Wurzelknoten, der im Speicher gehalten wird, solange das Programm läuft, sowie mehrere Overlay-Knoten, die bei Bedarf in den Speicher gelesen werden.

7.1 Einführung

ALINK produziert aus einem oder mehreren Eingabe-Files ein binäres Ausgabe-File. Die Eingabe-Files, auch Objekt-Files genannt, können Informationen über externe Symbole erhalten. Objekt-Files werden mit einem Assembler oder Compiler erstellt. Bevor der Linker das Ausgabe- oder auch ausführbare File erstellt, werden alle Bezüge auf Programmteile in Symbole umgewandelt.

Der Linker erzeugt außerdem, wenn das gewünscht wird, eine Cross-Referenz-Tabelle und ein Hilfs-File, »Link-Map« genannt. In den Linker ist ein sogenannter Overlay-Supervisor integriert. Mit diesem Programm können Module, die in einer Vielzahl von Programmiersprachen geschrieben wurden, zu einem Overlay-Programm zusammengebunden werden.

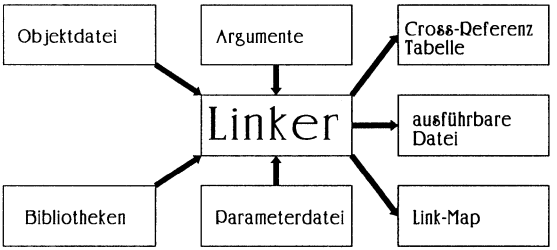


Bild 7.1: Das Prinzip eines Linkers

Der Linker kann seine Argumente auf zwei verschiedenen Wegen erhalten:

1. Aus der Befehlszeile. Die Informationen, die der Linker zur Arbeit braucht, werden als Argumente übergeben, wie bei CLI-Kommandos üblich.
2. Aus einem Parameter-File. Diese Alternative bietet sich an, wenn ein Programm mehrmals gelinkt werden soll. Alle Parameter werden dann in dem Parameter-File gespeichert. Beim Aufruf des Linkers werden alle Parameter aus dem File übernommen.

Die beiden Methoden können drei Arten von Input-Files verwenden:

1. Primär binärer Input:

Ein oder mehrere binäre Files bilden den Input zum Linker. Diese Files werden immer in das ladbare File übernommen, dieses Input-File muß immer vorhanden sein.

2. Overlay-Files:

Wenn Sie Ihr ladbares File im Overlay-Verfahren erstellen, ist das primäre Input-File das Basis-File, die nachgeladenen Files bilden den Rest der Programmstruktur.

3. Bibliotheks-Files:

Diese beziehen sich auf benötigten Code aus Bibliotheken, die der Linker automatisch einbindet. Dabei können sowohl im Betriebssystem vorhandene (resident Libraries) als auch als Bibliothek erstellte Files (scanned Libraries) eingebunden werden. Der Linker lädt eine solche Bibliothek nur, wenn ein externes Symbol aus der Bibliothek benötigt wird.

Der Linker arbeitet ein File in zwei Durchläufen (Passes) ab.

1. Im ersten Durchlauf werden alle Primär-, Bibliotheks- und Overlay-Files eingelesen sowie Code-Segmente und externe Symboltabellen definiert. Am Ende des ersten Durchlaufes werden, wenn gewünscht, die Cross-Referenz-Tabelle und die Link-Map erstellt.
2. Haben Sie ein Ausgabe-File angegeben, macht der Linker nun einen zweiten Durchlauf durch die Eingabe-Files. Zuerst wird das Primär-File in das Ausgabe-File kopiert. Dabei werden alle Referenzen aufgelöst. Dann werden auf gleichem Weg die Biblio-

theks-Files eingebunden. Beachten Sie bitte, daß Bibliotheks-Routinen in den Wurzel-Teil des Overlays eingehen. Dann werden die Daten für den Overlay-Supervisor erstellt und die Overlay-Files in das Ausgabe-File kopiert.

Im ersten Durchlauf, nachdem alle Primär- und Overlay-Files eingelesen sind, überprüft der Linker die Symboltabellen auf noch nicht zugeordnete Symbole. Findet er solche, liest er die von Ihnen angegebenen Bibliothek-Files. Der Linker markiert dann alle Code-Segmente, die Definitionen für diese nicht aufgelösten Symbolzuweisungen enthalten. Der Linker bindet nur die Bibliotheks-Code-Segmente ein, auf die sich die Symboltabelle bezieht.

7.2 Zur Anwendung des Linkers

Um den Linker einzusetzen, müssen Sie die Befehls-Syntax, seine Ein- und Ausgaben und die möglicherweise auftretenden Fehler kennen. Dieser Abschnitt vermittelt Ihnen diese Kenntnisse.

7.2.1 Syntax der Befehlszeile

Der Aufruf von ALINK hat folgende Form:

```
ALINK [FORM|ROOT] <files> [TO <file>] [WITH <file>] [VER <file>]
[LIBRARY|LIB <files>] [MAP <file>] [XREF <file>][WIDTH n]
```

Das Muster für die Schlüsselworte ist:

```
"FROM=ROOT,TO/K,WITH/K,VER/K,LIBRARY=LIB/K,MAP/K,XREF/K,WIDTH/K"
```

Dabei steht <file> jeweils für ein File, <files> für eine Liste von null oder mehr Files und n für eine Integer-Zahl. File-Namen in Listen müssen durch Kommata oder »+« getrennt werden.

Folgende Zeilen sind beispielsweise gültige Aufrufe von ALINK:

```
ALINK a
ALINK ROOT a+b+c+d MAP mapfile WIDTH 120
ALINK a,b,c TO ausgabe LIBRARY :flib/lib,obj/neulib
```

Die Liste der Files kopiert der Linker in der Reihenfolge wie angegeben in das Ausgabe-File. Die einzelnen Parameter haben folgende Bedeutung:

FROM — Gibt die Objekt-Files an, die primär eingelesen werden. Der Linker kopiert diese Files zu dem Lade-File, um die Overlay-Wurzel zu erstellen. Ein primäres File muß angegeben werden. (ROOT kann synonym zum FROM verwendet werden.)

TO — Gibt das Ausgabe-File an. Wird dieses File nicht angegeben, wird ein zweiter Linker-Durchlauf gestartet.

WITH — Bezeichnet Files, in denen die Parameter für den Linker gespeichert sind. Normalerweise wird hier nur ein File eingegeben, jedoch können auch mehrere, sich ergänzende Files angegeben werden. Parameter in der Befehlszeile setzen die Parameter in »files« außer Kraft. Die genaue Form dieser Files wird in Abschnitt 8.2.2 dieses Handbuches geschildert.

VER — Bezeichnet das File, in das der Linker seine Meldungen schreibt. Ist dieses File nicht angegeben, wird die Ausgabe zum normalen Ausgabe-Device – im allgemeinen zum Bildschirm – geleitet.

LIBRARY — Bezeichnet Bibliotheksdateien aus denen Funktionen in den Objektcode eingebunden werden. Der Linker bindet nur die Segmente (Funktionen) in den Objektcode ein, die für die ausführbare Datei (executable) benötigt werden. (LIB ist eine gültige Abkürzung für LIBRARY.)

MAP — Gibt die Ziel-Datei für die Link-Map an.

XREF — Gibt die Ziel-Datei für die Cross-Referenz-Tabelle an.

WIDTH — Definiert die Anzahl Zeichen pro Zeile, die für die Ausgabe der Link-Map und der Cross-Referenz-Tabelle verwendet werden. Dieser Parameter ist wichtig, wenn diese Ausgaben auf den Drucker geschrieben werden sollen.

7.2.2 WITH-Files

WITH-Files enthalten Parameter für den Linker. Sie werden diesen Befehl nur verwenden, wenn eine lange Sequenz von ALINK-Parametern häufiger verwendet werden soll oder die Namen der Files sehr Tippfehleranfällig sind.

Ein WITH-File besteht aus einer Reihe von ALINK-Schlüsselwörtern, je einem pro Zeile, mit den dazugehörigen Argumenten. Eine Zeile kann mit einem Semikolon abgeschlossen werden, der Linker ignoriert dann den Rest der Zeile. Nach dem Semikolon können Sie Kommentare eingeben. Leerzeilen werden ignoriert. An Schlüsselwörtern stehen zur Verfügung:

```
FROM/ROOT files
TO file
LIBRARY files
MAP [file]
XREF [file]
OVERLAY Struktur-Angabe
#
WIDTH zahl
```

Dabei steht »file« für ein einzelnes File, »files« für eine Liste von einem oder mehreren Files und »[file]« für ein optionales File. »zahl« muß eine ganze Zahl sein. Der Stern (*) kann zum Aufteilen langer Zeilen benutzt werden: Ein Stern am Ende einer Zeile teilt dem Linker mit, daß die folgende Zeile zu der mit dem Stern gehört. Geben Sie nach MAP oder XREF kein File an, werden die entsprechenden Tabellen zum File VER geschrieben und dadurch normalerweise auf den Bildschirm. Parameter, die Sie in der Befehlszeile einge-

ben, ersetzen Parameter zum gleichen Schlüsselwort in WITH-Files. So können Sie Standard-WITH-Files mit abweichenden Parametern nutzen. Geben Sie innerhalb des WITH-Files mehrere Parameter für ein Schlüsselwort an, wird das erste eingesetzt.

Auch in dem zweiten untenstehenden Beispiel trifft das übrigens zu, selbst wenn einem Schlüsselwort der Parameter Null zugewiesen wird.

Hier nun ein paar Beispiele zu WITH-Files und den dazugehörigen ALINK-Aufrufen:

```
ALINK WITH link-file
```

wobei »link-file« so aussieht:

```
FROM obj/haupt,obj/nächst
TO  bin/test
LIBRARY obj/lib
MAP
XREF x0
```

erzielt das gleiche Ergebnis wie:

```
ALINK FROM obj/haupt,obj/nächst TO bin/test LIBRARY
obj/lib MAP XREF x0
```

Der Befehl

```
ALINK WITH lkin LIBRARY ""
```

wobei »lkin« enthält:

```
FROM bin/prog,bin/subs
LIBRARY nag/fort.lib
TO linklib/prog
```

gibt das gleiche Ergebnis wie der Befehl:

```
ALINK FROM bin/prog,bin/subs TO linklib/prog
```

In dem obigen Beispiel ersetzt der Parameter ohne Wert in der Befehlszeile den Parameter im Link-File. Deshalb wird kein Bibliotheks-File eingebunden.

7.2.3 Fehler und andere Ausnahmebehandlungen

Während des Linker-Durchlaufes können verschiedene Fehler auftreten. Die meisten Meldungen erklären sich selbst und verweisen auf Fehler beim Öffnen der Files oder auf Fehler innerhalb der Parameter oder eines Binär-Files. Ein Fehler beendet meist den Link-Vorgang.

Einige Meldungen jedoch sind nur Warnungen. Die wichtigsten verweisen auf undefinierte oder mehrmals definierte Symbole. Der Linker-Durchlauf wird nach einer Warnung nicht abgebrochen. Wird im ersten Durchlauf ein undefiniertes Symbol entdeckt, erzeugt der Linker eine Warnung und gibt eine Liste solcher Symbole aus. Während des zweiten Durchlaufes werden die undefinierten Symbole auf die konstante Adresse NULL gesetzt.

Findet der Linker während des ersten Durchlaufes mehrere Definitionen für ein Symbol, so erzeugt er eine Warnung und ignoriert die zweite und jede weitere Definition. Diese Meldung wird nicht ausgegeben, wenn die Definition in einem LIBRARY-File auftaucht. Sie können also LIB-Funktionen einbinden, ohne einen Fehler zu erzeugen, auch wenn im Quell-Code und in einem LIB-File das gleiche Symbol verwendet wird. Ein ernsthafter Fehler tritt allerdings auf, wenn der Linker auf widersprüchliche Symbol-Zuweisungen stößt. Der Linkvorgang wird sofort abgebrochen.

Da der Linker nur die erste Definition eines Symbols übernimmt, ist es wichtig, daß Sie berücksichtigen, in welcher Reihenfolge Files eingelesen werden.

1. Primäre Eingabe-Files (FROM oder ROOT).
2. Overlay-Files.
3. LIBRARY-Files.

Innerhalb jeder Gruppe werden die Files in der im Argument angegebenen Reihenfolge eingelesen. Also haben Definitionen in FROM-Files die höchste und solche in LIBRARY-Files die niedrigste Priorität.

7.2.4 MAP und XREF

Die Link-Map, die der Linker am Ende des ersten Durchlaufes erzeugt, listet alle Code-Segmente, die im zweiten Durchlauf in dieser Reihenfolge in das ladbare File geschrieben werden.

Für jedes Code-Segment gibt der Linker eine Kopf-Zeile aus, bestehend aus dem Namen des Files, gekürzt auf acht Buchstaben, der Code-Segment-Referenznummer, dem Datentyp (Data, Code, BSS oder COMMON) und der Größe. Ist es ein Overlay-File, werden dazu noch die Overlay-Ebene und der Overlay-Rang ausgegeben.

Nach dem Kopf gibt der Linker jedes in dem Code-Segment definierte Symbol mit seinem Wert aus. Die Symbole werden nach ihrem Wert geordnet ausgegeben. Absolut definierten Symbolen wird ein »*« angehängt.

Der Wert des WIDTH-Schlüsselwortes gibt die Anzahl an Zeichen pro Zeile für diese Tabelle an. Ist der Wert sehr klein eingestellt, wird ein Symbol pro Zeile ausgegeben.

Die Cross-Referenz-Tabelle listet jedes Code-Segment. Die Kopfzeile ist dabei die gleiche wie bei MAP.

Der Kopfzeile folgt eine Liste der Symbole mit ihren Referenzen. Jede Referenz enthält zwei Integer-Zahlen, die den Versatz der Referenz und die Nummer des Code-Segmentes, auf das die Referenz weist, anzeigen. Die Nummer eines Code-Segmentes steht in seiner Kopfzeile.

7.3 Overlays

Das Overlay-System des Linkers und der Overlay-Supervisor bieten besonders Rechnerkonfigurationen mit wenig Speicher die Möglichkeit, große Programme auszuführen, ohne die Programm-Struktur direkt zu verändern.

Das vom Linker erzeugte ablauffähige Programm besteht auch beim Overlay-Modus aus einer Binär-Datei, die eine mit einem Baum vergleichbare Struktur besitzt. Dieses Programm enthält alle Code-Segmente und die »Einbau-Anleitung« für die Äste des Overlay-Baumes. Beim Laden des Programms wird das Hauptprogramm (zusammen mit den LIB-Segmenten und den COMMON-Blocks) – die Wurzel des Baumes – vom Loader in den Hauptspeicher geladen und verbleibt dort. Die einzelnen Segmente (sogenannte Overlay-Files), die die Äste unseres Baumes bilden, werden von dem Overlay-Supervisor, je nach Bedarf des Programmes, geladen beziehungsweise gelöscht. Sie sind angeordnet, wie in der Overlay-Direktive angegeben. Dieser Overlay-Supervisor wird beim Link-Vorgang automatisch mit eingebunden, wenn Overlays benutzt werden. Die Arbeit des Overlay-Supervisors bleibt für den Anwender und Programmierer unsichtbar.

7.3.1 Die OVERLAY-Direktive

Um die Struktur des Programm-»Baumes« zu definieren, wird die Direktive OVERLAY eingesetzt. Diese Direktive darf nur in WITH-Files angewandt werden. Die erste in einem WITH-File angegebene OVERLAY-Direktive wird ausgeführt.

Das Format für OVERLAY lautet:

```
OVERLAY
Xfiles
.
.
.
#
```

Die Overlay-Direktive kann sich über mehrere Zeilen erstrecken. Das Ende der Direktive wird mit »#« angegeben. Findet der Linker dieses Zeichen nicht, verwendet er alle Zeilen bis zur End-of-File-Marke als Argumente der OVERLAY-Direktive.

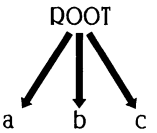
Jede Zeile nach OVERLAY definiert einen Zweig des Programm-Baumes und besteht aus einem Zähler »X« und einer Liste von Files.

Die Ebene eines Astes in der Höhe des Baumes wird mit keinem, einem oder mehreren »*«, als Zähler, definiert. Der Zähler »X«, der in der OVERLAY-Direktive angegeben wird, besteht also aus Null oder mehreren »*«-Sternen. Wird kein Stern eingegeben, gehört das Overlay-File zur Ebene der Wurzel. Die Overlay-Ebene eines Files errechnet sich durch: X+1. Zusätzlich zur Ebene hat jedes File noch einen Ordnungswert. Er gibt die Reihenfolge an, in der die Files eines Astes gelesen werden sollen. File 1 ist das erste File nach der Verzweigung. Es kann mehrere Äste mit gleicher Ebene und gleichem Ordnungswert geben. Sie müssen jedoch von verschiedenen Ursprungsästen ausgehen!

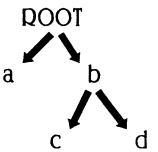
Während der Linker eine Overlay-Direktive bearbeitet, erinnert er sich der aktuellen Ebene und vergleicht die angegebene Höhe für jeden neuen Ast mit der vorhergehenden Nummer. Ist die neue Zahl kleiner, ist der neue Ast ein Abkömmling eines vorangehenden. Ist sie gleich, hat der neue Ast denselben Ursprung wie der letzte Ast. Ist die Zahl größer, ist der neue Ast ein direkter Abkömmling des letzten.

Hier einige Beispiele zur Verdeutlichung:

Direktive	Ebene	Ordinate
OVERLAY		
a	1	1
b	1	2
c	1	3
#		



Direktive	Ebene	Ordinate
OVERLAY		
a	1	1
b	1	2
*c	2	1
*d	2	2
#		



Direktive	Ebene	Ordinate
OVERLAY		
a	1	1
b	1	2
*c	2	1
*d	2	2
e	1	3
f	1	4
*g	2	1
*h	2	2
**i	3	1
**j	3	2
*k	2	3
l	1	5
#		

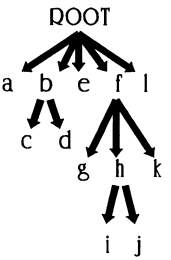


Bild 7.2: Unterschiedliche OVERLAY-Bäume

Während des Link-Vorganges werden die Overlay-Files, die Sie in der Direktive angegeben haben, Zeile für Zeile eingelesen. Die Reihenfolge wird in der Link-Map und der Cross-Reference-Tabelle festgehalten. Die Overlay-Ebenen und Ordnungszahlen werden dabei mit ausgegeben, so daß Sie die Struktur des Programmes jederzeit rückverfolgen können.

Die Ebenen und Ordnungszahlen beziehen sich ja auf den Ursprung in derselben Zeile. Bitte beachten Sie, daß alle oben mit den einzelnen Buchstaben bezeichneten Files auch ganze Listen von Files sein können.

Zum Beispiel:

```
ROOT bin/hauptaaa
OVERLAY
bin/hauptbbb,bin/hauptccc,bin/hauptddd
*bin/machreal
*bin/trbblock,bin/transint,bin/transr*
bin/transri
bin/codeaus
#
```

ergibt den im folgenden Bild gezeigten Programmbaum:

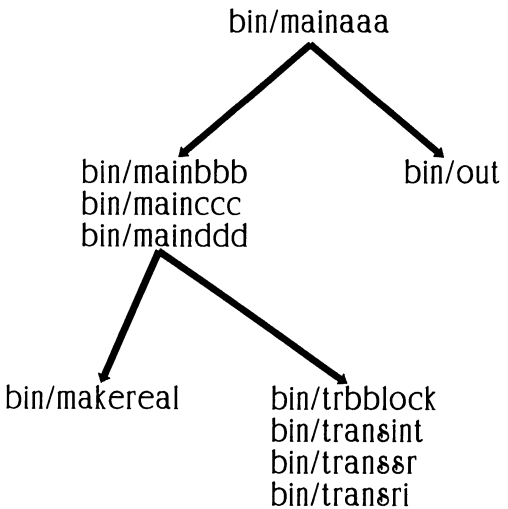


Bild 7.3: Ein Programmbaum

7.3.2 Symbol-Referenzen

Während ein Programm mit OVERLAY gelinkt wird, überprüft der Linker jeden Symbol-Verweis auf seine Zulässigkeit.

Nehmen wir an, daß der Verweis in der Verzweigung »V« und das auf das Segment verweisende Symbol in der Verzweigung »S« steht. Die Zuweisung ist zulässig, wenn eine der drei untenstehenden Bedingungen wahr ist.

1. »V« und »S« stehen in derselben Verzweigung.
2. »V« steht in einer nachgeordneten Ebene zu »S«.
3. »V« und »S« stehen in zwei verschiedenen Zweigen, aber auf gleicher Ebene.

Bezugnahmen der dritten Regel werden auch Overlay-Referenzen genannt. In diesem Fall startet der Loader den Overlay-Supervisor während des Programmablaufes. Der Supervisor überprüft dann, ob das Segment mit dem Symbol noch im Speicher steht. Wenn nicht, werden alle vorhandenen Segmente dieser Ebene und tiefer gelöscht und dann das mit dem Symbol eingeladen. Ein Overlay-Segment springt direkt zum aufrufenden Segment zurück, daher wird es nicht gelöscht, bis ein Segment höherer Ebene »darüber« geladen wird.

Angenommen, der Programm-Baum sieht wie folgt aus:

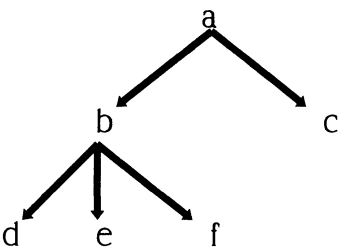


Bild 7.4: Ein einfacher Programmbaum

Lädt der Loader das Programm, steht zunächst nur Teil A im Speicher. Findet der Loader in A ein Symbol mit Bezug auf B, wird dieses Segment geladen und gestartet. Wenn B nun D aufruft, wird eine neue Verzweigung geladen. Verweist B auf A zurück, verbleiben die Segmente B und D im Speicher und brauchen nicht neu geladen zu werden, wenn auf sie verwiesen wird. Nehmen wir nun an, A ruft C auf. Nun löscht der Loader erst die nicht mehr benötigten Segmente und macht damit den Speicher für den neuen Teil frei. In diesem Fall also B und D. Erst wenn diese Teile gelöscht sind, kann der Loader C laden.

Wird ein Segment ausgeführt, sind alle darüberliegenden Segmente und eventuell einige nachgeordnete Segmente im Speicher.

7.3.3 Warnungen und Hinweise

Der Linker nimmt an, daß Zugriffe in andere Teile des Overlays Sprünge oder Unterprogramm-Aufrufe sind. Diese Verzweigungen werden mit dem Overlay-Supervisor gesteuert. Sie sollten deshalb keine Symbole, die Sie für den Zugriff auf Daten verwenden, als Overlay-Symbole verwenden.

Der Linker erteilt jedem Segment, das eine Overlay-Zuweisung enthält, eine Overlay-Nummer. Diese Nummer, die immer größer oder gleich Null ist, wird benutzt, um ein Label zu konstruieren, das dem Einsprungpunkt in dieses Overlay entspricht. Das Label hat

die Form »OVLYnnnn« wobei »nnnn« die eben erwähnte Overlay-Nummer ist. Dieses Format für Labels sollten Sie deshalb nicht selbst verwenden.

Der Linker zieht alle Sektionen mit demselben Sektions-Namen zu einem Code-Segment zusammen. Damit können diese Segmente einfacher in einem Stück in den Speicher eingeladen werden.

7.4 Weiterführende Hinweise

7.4.1 Überblick über die Hunk-Overlay-Tabelle

Werden Overlays verwendet, erzeugt der Linker zuerst ein langes File aus allen Objektfiles, das eine Folge von Modulen mit relozierbarem Code enthält. Die Hunk-Overlay-Tabelle enthält eine Datenstruktur, die jedes Modul und deren Beziehung zueinander beschreibt.

Erstellen Sie ein Programm, das Overlays verwenden soll, müssen Sie sich vor Augen halten, wie der Overlay-Supervisor die Interaktionen zwischen den einzelnen Segmenten steuert. Zuerst sollten Sie den Baum erstellen, aus dem hervorgeht, in welcher Beziehung die einzelnen Module zueinander stehen. Diese Beziehungen teilen Sie dann dem Linker mit.

Die Hunk-Overlay-Tabelle besteht aus acht Langworten, jedes von ihnen beschreibt einen Overlay-Knoten und ist Teil des Overlay-Files. Jeder Acht-Wort-Eintrag enthält folgende Daten:

```
Wert  Funktion       Beschreibung
LONG  seekOffset;    /* Wo im File befindet sich dieser Knoten */
LONG  dummy1;        /* Ein Wert von 0 ... zusammengehöriger
                        Punkte */
LONG  dummy2;        /* Ein Wert von 0 ... zusammengehöriger
                        Punkte */
LONG  level;         /* Ebene im Baum */
LONG  ordinate;      /* Anzahl an Punkten auf dieser Ebene */
LONG  firstHunk;     /* Modul-Nummer des ersten Knotens zu
                        diesem Modul */
LONG  symbolHunk;    /* die Nummer des Moduls, zu dem dieses
                        Symbol gehört */
LONG  symbolOffsetX; /* (Offset+4), wobei Offset der Offset
                        des Symbol-Hunk zu dem jeweiligen Symbol-
                        Eintrag ist.*/
```

Jeder dieser Einträge wird in den nun folgenden Abschnitten näher beschrieben.

7.4.1.1 Einen Overlay-Baum planen

Gehen wir davon aus, daß die folgenden Module an der Overlay-Struktur beteiligt sind: haupt, a, b, c, d, e, f, g, h, i und j. haupt ruft die Module a, b, c und d auf, und jedes dieser Module kann haupt aufrufen. Weiterhin nehmen wir an, daß Modul e von den Modulen haupt, a, b und c aufgerufen werden kann, aber keine Beziehung zum Modul f hat. Soll also eine Routine aus e aufgerufen werden, müssen weiterhin die Module a, b, c und d im Speicher zur Verfügung stehen. Routine f verhält sich wie e, das heißt, alle Module außer e müssen im Speicher bereitstehen, wenn f von a, b, c oder d aufgerufen wird. Der Overlay-Supervisor kann also die Routinen e und f nacheinander in den gleichen Bereich des Speichers schreiben, da die eine die andere zur Funktion nicht benötigt.

Nehmen wir nun weiter an, Modul g belegt denselben Speicherbereich wie a, b, c und d und auch die Routinen h, i und j passen in denselben Platz, so ergibt sich folgende Struktur für die Basis unseres Baumes:

```
Haupt (Wurzel-Ebene des Baumes)
                                |
     |
                          a, b, c, d (1,1)
   g (1,2)
                          |     |        |
  |      |
e(2,1)  f (2,2)       h (2,1)  i (2,2)  j (2,3)
```

Nun haben wir nicht nur den Baum aufgezeichnet, sondern auch gleich noch seine Äste benannt. Diese Hunk-Overlay-Nummern finden sich auch in der Hunk-Overlay-Tabelle und bezeichnen die Knoten, denen sie zugeordnet sind.

Aus der Beschreibung können Sie ersehen, daß alle Routinen aus den Modulen a, b, c und d immer gleichzeitig im Speicher stehen, wenn aus haupt eine dieser Routinen aufgerufen wird, da sie vom Linker alle im gleichen Knoten untergebracht sind. Während a–d im Speicher stehen, kann der Linker File e immer gleich einlesen und reinitialisieren (bei jedem Aufruf neu!), wenn eine Routine daraus aufgerufen wird. Wird aus einem der Files a–d das Modul f benötigt, ersetzt der Linker den Inhalt des Speichers, in dem e steht, durch f und initialisiert dieses Modul. So sind die Files a–d die erste Ebene im Overlay-Baum. Die Routinen in e und f stellen Ebene 2 dar, alle Module aus Ebene 1 müssen im Speicher stehen, um sie verfügbar zu machen.

Eine Routine kann natürlich nur dann auf andere Module zugreifen, wenn dieses andere Modul im Speicher steht oder im direkt nachgeordneten Knoten eingereiht ist. haupt kann also nicht auf e zugreifen, ohne vorher a–d eingelesen zu haben. e kann jedoch auf Routinen in haupt zugreifen, da haupt immer im Speicher steht, wenn e aufgerufen wird.

Beachten Sie auch, daß jeder Hauptast und seine Knoten wieder mit 1 beginnend numeriert werden.

7.4.1.2 Beschreibung des Baums

Der Baum wird erzeugt, indem Sie dem Linker dessen Struktur mitteilen. Die numerischen Werte, gleich denen in unserer Zeichnung oben, werden vom Linker nacheinander zugeteilt und erscheinen in der Hunk-Node-Tabelle.

Hier nun die Sequenz, die den oben gezeigten Baum entstehen läßt:

```
OVERLAY
a,b,c,d
*e
*f
g
*h
*i
*j
```

Diese Beschreibung zeigt dem Linker, daß a, b, c und d Teile eines Knotens in einer gegebenen Ebene (in diesem Fall Ebene 1) sind. Der Stern (*) vor e und f zeigt an, daß diese Module eine Ebene unter a–d liegen. Sie können also nur durch a–d oder von im Baum weiter oben stehenden Modulen aufgerufen werden. Der Name g trägt keinen Stern, damit steht er auf derselben Ebene wie a–d. Somit erfährt der Linker, daß entweder a–d oder g im Speicher sein können, nicht jedoch alle Files. h-j stehen eine Ebene unter g.

Die obenstehenden Abschnitte haben den Ursprung des Overlay-Baumes und die Reihenfolge der ersten Files in der Hunk-Overlay-Symbol-Tabelle erklärt.

seekOffset

Der erste Wert für jeden Knoten in der Overlay-Tabelle ist der seekOffset. Wie bereits gesagt, bildet der Linker aus allen Overlay-Knoten ein großes File. Der Wert des seek-Offset wird der seek-Routine übergeben (mit den Parametern file, byte_offset) und ergibt den Pointer auf das erste Byte des hunkHeader des Knotens.

initialHunk

Der Wert in initialHunk wird vom Overlay-Supervisor verwendet, wenn ein Knoten überschrieben wird. Er verweist auf das Modul, das geladen sein muß, um den Knoten zu laden, der das Symbol enthält. Wird eine Routine aufgerufen, die zu einem anderen Knoten oder einer anderen Ebene gehört (wenn es sich nicht um ein direkt dazugehöriges, ein direkt nachgeordnetes oder ein direkt davorstehendes File handelt), muß der Speicher freigemacht werden. Dazu werden alle nicht benötigten Module gelöscht und Raum für das Modul mit dem angegebenen Symbol gemacht.

SymbolHunk und SymbolOffsetX

Diese Einträge zu den Symbolen werden vom Overlay-Supervisor verwendet, um den aktuellen Einsprungpunkt eines geladenen oder zu ladenden Moduls zu finden. SymbolHunk zeigt auf das Modul, in dem das Symbol zu finden ist, SymbolOffsetX-4 zeigt den Offset vom Beginn des Moduls an, bei dem der Einsprungpunkt im Augenblick liegt.

7.5 Ladbare Dateien (Load-Files)

Das Format eines Load-Files (des Ausgabe-Files des Linkers) ist ähnlich dem eines Objekt-Files. Es enthält eine Anzahl von Modulen mit gleichem Format wie die Module in Objekt-Files. Der hauptsächliche Unterschied ist, daß ein Load-File niemals einen Block mit externen Symbol-Informationen enthält. Alle Symbolreferenzen sind aufgelöst, die Programmeinheit-Informationen werden nicht übernommen. Ein einfaches Load-File (nicht im Overlay erstellt) enthält einen Header-Block, der die Anzahl der Module innerhalb des Load-Files und alle residenten Libraries angibt. Diesem Block folgen die Module, die jeweils aus mehreren Eingabe-Modulen mit gleichem Namen bestehen können. Diese ganze Struktur wird als ein Knoten behandelt. Zusätzlich kann im Load-File auch die Overlay-Information enthalten sein. In diesem Fall folgt dem primären Knoten die Overlay-Tabelle. Ein spezieller Break-Block trennt die Overlay-Knoten voneinander. Diese Overlay-Struktur kann wie folgt dargestellt werden, wobei optionale Teile mit einem Stern bezeichnet sind.

- `Primärer Knoten`
- `Overlay- Tabelle (*)`
- `Overlay-Knoten, getrennt mit Break-Blöcken (*)`

Die Relokations-Blöcke innerhalb der Module sind immer vom Typ hunk_reloc32. Die Relokation geschieht während des Ladens. Dabei werden nicht nur die Relokationen mit hunk_reloc32, sondern auch die Auflösung der externen Symboltabelle durchgeführt.

Jede externe Referenz im Objekt-File wird wie folgt behandelt: Der Linker durchsucht den primären Input nach passenden externen Definitionen. Findet er diese nicht, durchforstet er die Scanned Libraries und bindet die gesamte Programmeinheit, in dem die Definition steht, in das Load-File ein. Dadurch können wieder externe Referenzen auftreten. Am Ende des ersten Durchlaufes kennt der Linker alle externen Definitionen und alle Module, die dem Programm angehören sollen. Im zweiten Durchlauf werden die Langworte der externen Referenzen so verändert, daß sie den benötigten Offset zu dem Modul herstellen, in dem das Symbol definiert ist. Wenn alle Module geladen sind, wird im Relokations-Block ein Eintrag erzeugt, der die Basisadresse jedes Moduls enthält, das das Symbol definiert. Dieses Verfahren gilt auch für residente Libraries.

Bevor der Lader diese Modul-übergreifenden Referenzen auflösen kann, muß er die Anzahl und Größe aller Module des jeweiligen Knotens kennen. Der weiter unten beschriebene Header-Block enthält diese Informationen. Das Load-File enthält auch even-

tuell Overlay-Informationen in einem speziellen Block, der Overlay-Tabelle. Break-Blöcke trennen die einzelnen Overlay-Knoten.

7.5.1 hunk_header (1011/3F3)

Dieser Block enthält die Informationen über die Anzahl und Länge aller zu ladenden Module. Ebenso sind die Namen aller residenten Libraries, die für diesen Knoten geöffnet werden müssen, enthalten. Das Format von hunk_header ist in Bild 7.5 dargestellt. Der erste Teil des Blocks enthält die Namen aller zu öffnenden residenten Libraries, die in diesen Knoten eingebunden werden sollen. Jeder Name besteht aus einem Langwort mit der Länge des Namens der residenten Library und dem Namen selbst in Langworten, aufgefüllt bis zur nächsten Langwortgrenze mit dem Wert Null. Die Liste der Namen endet mit einem Langwort mit dem Wert Null. Die residenten Libraries werden in der angegebenen Reihenfolge geöffnet.

Wird ein primärer Knoten geladen, erstellt der Lader eine Tabelle im Arbeitsspeicher, in der die Reihenfolge aller geladenen Module gespeichert wird. Die Tabelle muß groß genug sein, um alle Module, auch die in den Overlays, unterzubringen. In diese Tabelle wird auch eine Kopie der Modul-Tabelle zusammen mit allen residenten Libraries geschrieben. Das nächste Langwort im Header enthält die Größe dieser Tabelle (Anzahl aller Module plus 1) in einem Langwort. Das nächste Langwort F verweist auf den ersten Wert in der Modul-Tabelle, die vom Lader benutzt werden soll. Für einen primären Knoten, der keine residenten Libraries eingebunden bekommt, ist dieser Wert Null; andernfalls enthält er die Anzahl von Modulen in den residenten Libraries. Der Lader kopiert diese Einträge der Modul-Tabelle zusammen mit der Library, gefolgt vom Aufruf der Funktion OpenLibrary(). Bei einem Overlay-Knoten ist dieser Wert die Anzahl der Module in allen residenten Libraries und die Anzahl aller bereits in höherstehenden Knoten geladenen Module.

Das nächste Langwort L zeigt auf das letzte zu ladende Modul in diesem Aufruf des Laders. Die Gesamtanzahl der geladenen Module ist dann L–F+1.

```
hunk_header
N1
N1 Langworte des Namens
...
N2
N2 Langworte des Namens
...
0
Größe der Tabelle
Erstes Modul F
Letztes Modul L
L-F+1 Größenangaben
...
```

Bild 7.5: hunk_header (1010/3F3)

Der Header-Block enthält dann L–F+1 Langworte. Jedes zeigt die Größe des jeweiligen Moduls an. Damit wird der Lader in die Lage versetzt, den gesamten benötigten Speicherplatz zu belegen, bevor das erste Modul geladen wird. Es steht dann genug Speicher zur Verfügung, um alle Symbole aufzulösen. Ist ein hunk_bss mit einer Größe Null vorhanden, verwendet der Lader eine Systemvariable als Größe dieses Moduls, wie in der Beschreibung von hunk_bss erläutert.

7.5.2 hunk_overlay (1013/3F5)

Der Block mit der Overlay-Tabelle zeigt dem Lader an, daß ein Programm im Overlay-Verfahren erstellt werden soll, und enthält alle dazu nötigen Informationen. Trifft der Lader auf diesen Block, bearbeitet er die Tabelle und kehrt dann zur weiteren Programm-Bearbeitung zurück. Dabei bleiben alle Input-Kanäle geöffnet. Grafisch dargestellt sieht hunk_overlay wie folgt aus:

```
hunk_overlay
Größe der Tabelle
M+2
M+1
Nullen
...
Overlay-Daten-Tabelle
...
```

Bild 7.6: hunk_overlay (1013/3F5)

Das erste Langwort ist die obere Grenze der gesamten Overlay-Tabelle in Langworten.

M ist die maximale Anzahl von Verzweigungen im Overlay-Baum; die Wurzel hat den Wert Null. Die nächsten M+1 Worte bilden das Gerüst der Overlay-Tabelle.

Der Rest des Blocks ist die Overlay-Datentabelle, eine Reihe von 8-Wort-Einträgen; eines für jedes Overlay-Symbol. Wird als maximale Overlay-Nummer Null eingesetzt, ist die Größe der Overlay-Datentabelle (0+1)*8, da die erste Overlay-Nummer Null ist. Die gesamte Größe der Overlay-Tabelle ist dann (0+1)*8+M+1.

7.5.3 hunk_break (1014/3F6)

Ein Break-Block zeigt das Ende eines Overlay-Knotens an. Er besteht aus einem Langwort mit dem Wert hunk_break.

7.6 Beispiele

Der folgende Abschnitt zeigt anhand kleiner Code-Segmente, wie Linker und Lader externe Symbole bearbeiten:

```
                    IDNT    A
                    XREF    BILLY, JOHN
                    XDEF    MARY
```

* Das nächste Langwort soll reloziert werden

```
0000' 0000 0008 DC.L    FRED
0004' 123C 00FF MOVE.B  #$FF,D1
0008' 7001      FRED    MOVEQ   #1,D0
```

*Externer Einsprung

```
000A' 4E71      MARY    NOP
000C' 4EB9 0000 0000            JSR BILLY
```

*Externer Aufruf

```
0012' 2239 0000 0000    MOVE.L  JOHN,D1
```

*Externe Referenz

```
                    END
```

erzeugt folgendes Objektfile:

```
hunk_unit
00000001        Größe in Langworten
41000000        Name, zum Langwort aufgefüllt mit Nullen
hunk_code
00000006        Größe in Langworten
00000008 123C00FF 70014E71 4EB90000 00002239 00000000
hunk_reloc32
00000001        Nummer in Modul 0
00000000        Modul 0
00000000        Offset zur Relokation
00000000        Null zum Ende
hunk_ext
01000001        XDEF, Größe: ein Langwort
4D415259        MARY
0000000A        Offset der Definition
81000001        XREF, Größe: ein Langwort
4A4F484E        JOHN
00000001        Anzahl der Referenzen
00000014        Offset der Referenzen
81000002        XREF, Größe: zwei Langworte
42494C4C        BILLY
59000000        (Aufgefüllt mit Nullen)
```

```
00000001        Anzahl der Referenzen
0000000E        Offset der Referenzen
00000000        Ende des externen Blocks
hunk_end
```

Das entspricht dem folgenden Programm:

```
                IDNT    B
                XDEF    BILLY, JOHN
                XREF    MARY
0000' 2A3C AAAA AAAA    MOVE.L  #$AAAAAAAA,D5
* Externer Einsprung
0006' 4E71      BILLY   NOP
* Externer Einsprung
0008' 7201      JOHN    MOVEQ   #1,D1

* Aufruf der externen Referenz

000A' 4EF9 0000 0000    JMP     MARY

                END
```

Der dazugehörige Output-Code sieht so aus:

```
hunk_unit
00000001        Größe in Langworten
42000000        Name des Teiles
hunk_code
00000004        Größe in Langworten
2A3CAAAA AAAA4E71 72014EF9      00000000
hunk_ext
01000001        XDEF, Größe: ein Langwort
4A4F484E        JOHN
00000008        Offset der Definition
01000002        XDEF, Größe: zwei Langworte
42494C4C        BILLY
59000000        (Aufgefüllt mit Nullen)
00000006        Offset der Definition
81000001        XREF, Größe: ein Langwort
4D415259        MARY
00000001        Anzahl der Referenzen
0000000C        Offset der Referenzen
00000000        Ende des externen Blocks
hunk_end
```

Nachdem der Linker das Ganze bearbeitet hat, erhalten Sie folgendes Load-File:

```
hunk_header
00000000        Kein Modul-Name
00000002        Größe der Modul-Tabelle
00000000        Erstes Modul
00000001        Letztes Modul
00000006        Länge von Modul 0
00000004        Länge von Modul 1
hunk_code
00000006        Länge des Codes in Langworten
00000008 123C00FF 70014E71 4EB90000 00062239 00000008
hunk_reloc32
00000001        Nummer in Modul 0
00000000        Modul 0
00000000        Offset der Relokation
00000002        Nummer in Modul 1
00000001        Modul 1
00000014        Offset der Relokation
0000000E        Offset der Relokation
00000000        Null zum Ende
hunk_end
hunk_code
00000004        Länge des Codes in Langworten
2A3CAAAA AAAA4E71 72014EF9 0000000A
hunk_reloc32
00000001        Nummer in Modul 0
00000000        Modul 0
0000000C        Offset der Relokation
00000000        Null zum Ende
hunk_end
```

Wird dieser Code vom Lader in den Speicher gebracht, liest er den Header-Block ein und erstellt eine Modul-Tabelle von zwei Langworten. Dann wird eine Systemroutine aufgerufen, die zwei Bereiche Speicher von sechs und vier Langworten Größe bereitstellt. Nehmen wir an, diese Bereiche beginnen bei den Speicherstellen 3000 und 7000, so enthält die Modul-Tabelle diese beiden Werte.

Der Lader liest das erste Modul und schreibt den Code nach 3000; dann wird die Relokation durchgeführt. Der erste Begriff bezieht sich auf Modul 0, also wird 3000 zu dem Langwort von Offset 0 addiert. Der Wert wird dann von 00000008 nach 00003008 gebracht. Der zweite Begriff bezieht sich auf Modul 1. Obwohl noch nicht geladen, wissen wir, daß diese Werte nach Speicherplatz 7000 gebracht werden sollen. Deshalb wird es zu den Werten in den Speicherstellen 300E und 3014 addiert. Beachten Sie, daß der Linker die Offsets 00000006 und 00000008 bereits in die Referenz in Modul 0 eingesetzt hat, deshalb entsteht der korrekte Offset zur Definition in Modul 1. Diese Langworte bestimmen

das Ende der externen Referenzen und enthalten nun die Werte 00007006 und 00007008, die korrekt sind, sobald das zweite Modul geladen ist.

Genauso lädt der Lader das zweite Modul nach Speicherstelle 7000. Die Informationen zur Relokation ändern das Langwort bei 700C von 0000000A (dem Offset von MARY im ersten Modul) nach 0000300A (der Adresse von MARY im Speicher – im zweiten Modul).

Der Lader bearbeitet Referenzen auf residente Libraries genauso. Er kopiert lediglich nach dem Öffnen der Library die Adressen der Module, aus denen die Library besteht, an den Anfang der Modul-Tabelle. Danach werden alle Referenzen, die auf die Library verweisen, korrigiert, indem er die Basisadresse des jeweiligen Moduls aufaddiert, um den richtigen Offset zu erhalten.

7.7 Fehlercodes und -meldungen

Diese Fehler werden sehr selten auftreten. Wenn doch, kann der Fehler auch am Compiler liegen. Sie sollten trotzdem überprüfen, ob Ihre Programme einwandfrei sind (z.B. muß ein Eingabe-Programm einen Einführungsteil enthalten, das dem Linker mitteilt, daß ein Programm zu erwarten ist).

Unzulässige Objektmodule

```
2       Overlay-Symbol falsch angewandt
3       Symbol falsch angewandt
4       COMMON falsch angewandt
5       Overlay-Zuweisung nicht korrekt
6       Overlay-Zuweisung nicht Null
7       Externen Block falsch verschoben
8       bss falsch verschoben
9       Programmteil falsch verschoben
10      Falscher Offset bei 32-Bit-Verschiebung
11      Falscher Offset bei 16/8-Bit-Verschiebung
12      Falscher Offset bei 32-Bit Zuweisung
13      Falscher Offset bei 16/8-Bit-Zuweisung
14      End-of-file nicht gefunden
15      Hunk.end nicht gefunden
16      Unzulässiges Fileende
17      File vorzeitig beendet
18      File vorzeitig beendet
```

Interne Fehler

```
19      Unzulässiger Typ in Hunk-Liste
20      Interner Fehler beim Libraryaufruf
21      Falsches Argument zu Free-Vektor
22      Symbol im zweiten Durchlauf nicht definiert
```

7.8 ATOM (Alink Temporary Object Modifier)

In diesem Abschnitt wird das ATOM-Hilfsprogramm, das vor dem Linker ALINK zum Einsatz kommt, erklärt. Außerdem werden seine Entstehungsgeschichte, die Art der Implementierung und Alternativen zu seinem Einsatz erläutert.

7.8.1 Das »Problem«

Programmierer müssen in der Lage sein, zu bestimmen, daß Teile ihres Programmes in den Chip-RAM (die ersten 512 Kbyte des RAM) geladen werden, damit die Custom-Chips darauf zugreifen können. Sie möchten aber auch, daß diese speziellen Daten wie andere Daten behandelt werden können, daher müssen sie normal eingebunden und geladen werden.

7.8.2 Die bisherige Lösung

Der bisher benutzte Weg zur Lösung dieses Problems sah so aus:

Zuerst wurde Alloc-Mem mit gesetzten Chip-RAM-Bits aufgerufen. Dann wurden die Daten von der Speicherstelle, in die sie geladen wurden, in den Chip-RAM kopiert. Auf diese Weise waren die Daten zweimal im Speicher vorhanden: einmal die ursprünglich geladenen und dann die in den ersten 512 Kbyte des Speichers.

Als andere Lösung blieb nur, dem Anwender zu sagen, daß das Programm in Rechnern mit mehr als 512 Kbyte nicht läuft. Aber das ist eine ziemlich unbefriedigende Lösung.

7.8.3 Die Lösung durch ATOM

1. Kompilieren oder assemblieren Sie wie gewohnt.
2. Bearbeiten Sie das Object-File mit dem Post-Prozessor ATOM. ATOM arbeitet interaktiv mit dem Anwender und fragt jedes Modul des Object-Files ab. Alle angegebenen werden im Typ als für CHIP-Speicher geändert.
3. Der Linker kennt nun neben den schon bekannten Modul-Typen hunk_code, hunk_data und hunk_bss noch sechs weitere Modul-Typen:

```
hunk_code_chip  = hunk_code     + Bit 30 gesetzt
hunk_code_fast  = hunk_code     + Bit 31 gesetzt
hunk_data_chip  = hunk_data     + Bit 30 gesetzt
hunk_data_fast  = hunk_data     + Bit 31 gesetzt
hunk_bss_chip   = hunk_bss      + Bit 30 gesetzt
hunk_bss_fast   = hunk_bss      + Bit 31 gesetzt
```

Der Linker bereitet alle Modul-Typen zur Bearbeitung mit dem Loader vor (und bindet sie zusammen, falls erforderlich). Der Loader verwendet die Informationen aus dem Modul-Header während des Ladens.

Beachten Sie, daß hunk_code ausführbare Maschinensprache-Elemente, hunk_data initialisierte Daten und hunk_bss nicht initialisierte Daten (Felder, Variablendefinitionen und so weiter) enthalten.

4. Der Loader lädt nun die Module entsprechend ihrem Typ in den jeweiligen Speicherbereich.
5. Die alten Versionen des Loaders interpretieren die neuen Modul-Typen als sehr lange Files und verweigern die Arbeit. (Fehler 103; nicht genug Arbeitsspeicher.)

7.8.4 Zukünftige Lösungen

Assembler und C-Compiler von Lattice werden voraussichtlich in naher Zukunft modifiziert werden, so daß diese Programme die neuen Modul-Typen bereits unter Programm-Kontrolle erstellen (siehe Lattice C 5.0 – BLINK).

7.8.5 Das bewirken die neuen Bits

Die Modul-Größe ist ein Langwort, das die Anzahl der Langworte im Modul enthält. Für die in nächster Zukunft zu erwartenden Rechner mit 32-Bit-Adreßbus wurden die beiden höchstwertigen Bits freigehalten. Diese Bits wurden für ATOM folgendermaßen umdefiniert (Bit 31 heißt auch MEMF_FAST und Bit 32 MEMF_CHIP):

Bit 31	Bit 32	Bedeutung
0	0	Ist kein Bit gesetzt, wird der gerade zur Verfügung stehende Speicher belegt. Vorzug wird FAST-Memory gegeben. Diese Lösung ist aufwärtskompatibel.
1	0	Der LOADER muß FAST-Memory belegen oder einen Fehler ausgeben.
0	1	Der LOADER muß CHIP-Memory belegen oder einen Fehler ausgeben.
1	1	Sind beide Bits gesetzt, folgt diesem Langwort ein weiteres mit zusätzlichen Informationen. Diese Funktion ist noch nicht genutzt, sondern dient zukünftigen Erweiterungen.

7.8.6 Wirkung der Neuerung

Programme, die nicht mit den neuen Optionen assembliert oder kompiliert wurden, laufen (oder auch nicht) wie gewohnt. Das heißt aber auch, daß sie mit der Speichererweiterung in Konflikt kommen, wenn sie nicht sauber programmiert worden sind. Die alte Lösung steht weiterhin zur Verfügung.

Programmentwicklung kann (auch mit einer Maschine mit 512 Kbyte) wie gewohnt durchgeführt werden. Erst wenn das Programm nahezu fertig ist, sollte ATOM benutzt und dann das Programm auf einem Amiga mit mehr als 512 Kbyte RAM ausgetestet werden. Diese Lösung funktioniert auf allen drei Entwicklungs-Rechnern (Amiga, Sun und IBM).

Die mit ATOM entwickelten Programme sind unter V1.0 von Kickstart (auf dem Amiga 1000) nicht lauffähig. Hier wird der Fehler 103 (nicht genug Arbeitsspeicher) gemeldet. OMD und DumpObj in den Versionen V1.0 oder früher vertragen sich ebenfalls nicht mit ATOM-bearbeiteten Files.

7.8.9 Erforderliche Software

Zur Arbeit mit ATOM brauchen Sie folgende, zur Version 1.1 kompatible Software in der angegebenen oder einer späteren Version:

```
ATOM      (Version 1.0 oder später)
Alink     (Version 3.30 oder später)
Kickstart (Release 1.1 oder später) für den Amiga 1000.
DumpObj   (Version 2.1) (brauchen Sie nur, wenn Sie mit ATOM
          bearbeitete Programme inspizieren wollen)
```

7.8.10 Syntax der ATOM-Anweisung

Die Syntax für den Aufruf von ATOM ist:

```
ATOM <einfile> <ausfile> [-I]
```

oder

```
ATOM <einfile> <ausfile> [-C[C|D|B]] [-F[C|D|B]] [-P[C|D|B]]
```

Dabei stehen:

<einfile>	für das Objekt-File, das gerade assembliert oder kompiliert oder auch bereits mit ATOM behandelt wurde. (Ein File kann mehrmals »ATOMisiert« werden.)
<ausfile>	für das Ziel-File von ATOM
–I	für interaktiven Ablauf
–C	für ändere Speicherbereich nach CHIP
–F	für ändere Speicherbereich nach FAST
–P	für ändere Speicherbereich nach PUBLIC (der Typ Speicher, der gerade zur Verfügung steht)
–C	für ändere nur CODE-Module
–D	für ändere nur DATA-Module
–B	für ändere nur BSS-Module

7.8.11 Beispiele für ATOM-Aufrufe

Beispiel 1
In den meisten Fällen braucht der Code des Programmes nicht im Chip-RAM zu stehen. Der Blitter braucht nur den Inhalt von DATA- und BSS-Modulen. Die folgende Anweisung trägt dem Rechnung. Alle Daten und Definitionen werden in den Chip-RAM geladen, der eigentliche Code des Programmes kommt ins Public-RAM. Geben Sie dazu ein:

```
ATOM einfile.obj ausfile.obj -pc -cdb
```

Beispiel 2
Um alle Module in den Chip-RAM zu laden, tippen Sie:

```
ATOM einfile.obj ausfile.obj -c
```

Beispiel 3
Um alle DATA-Module in den Chip-RAM zu laden, tippen Sie nun:

```
ATOM meinfile.o meinfile.at.o -cd
```

Beispiel 4
Hier ein Beispiel für den interaktiven Dialog mit ATOM. Die Eingabe des Anwenders wird in Kleinbuchstaben, die Ausgabe des Rechners in Großbuchstaben abgedruckt. In diesem Beispiel soll das CODE-Modul nach FAST, das DATA-Modul nach CHIP geladen werden. BSS-Module erscheinen nicht. Beachten Sie, daß zu Beginn HELP angefordert wird.

```
2> atom von.o von.set.o -i
AMIGA OBJECT MODIFIER V1.0
UNIT NAME FROM                ; Eingabefile
HUNK NAME NONE                ; kein Modulname
HUNK TYPE CODE                ; Modultyp ist CODE MEMORY
ALLOCATION PUPLIC             ; Speicherbereich nicht extra angegeben
DISPLAY SYMBOLS [Y/N] y       ; Symbole aufzeigen? ja
_basis
_xcovf.
_CXD22..
_druckf.
_haupt...
MEMORY TYPE? [F|C|P] ?        ; ? fordert Hilfe an
PLEASE ENTER F FOR FAST       ; Speicherarten
             C FOR CHIP
             P FOR PUPLIC
             Q TO QUIT        ; Programm beenden, kein Ausgabe-File
                                wird erstellt
             W TO WINDUP      ; Verändert den Rest des Files nicht,
                                übernimmt ihn in das Ausgabe-File
             N FOR NEXT HUNK  ; springt unmittelbar zum nächsten
                                Modul
```

```
MEMORY TYPE? [F|C|P] f
UNIT NAME 0000
HUNKNAME NONE
HUNK TYPE DATA        ; Daten-Modul
MEMORY ALLOCATION PUPLIC
DISPLAY SYMBOLS? [Y/N] n
MEMORY TYPE? [F|C|P] c
UNIT NAME 0000
HUNKNAME NONE
HUNK TYPE BSS
MEMORY ALLOCATION PUPLIC
DISPLAY SYMBOLS? [Y/N] y      ; kein BSS-Modul!
MEMORY TYPE? [F|C|P] p
2>_
```

7.8.12 Fehlermeldungen

Fehler: *Bad args* (fehlerhafte Argumente)

```
a) Eine Option beginnt nicht mit dem -
b) Falsche Anzahl an Parametern
c) Dem - folgt kein I, C, F oder P
d) -x und -I zusammen und so weiter
```

Fehler: *Bad infile* (unzulässiges Eingabefile)

```
File wurde nicht gefunden.
```

Fehler: *Bad outfile* (unzulässiges Ausgabefile)

```
Ausgabe-File kann nicht erstellt werden (Diskette voll,
Schreibschutz aktiv und so weiter).
```

Fehler: *Bad Type* ## (falscher Modul-Typ)

```
ATOM hat ein Modul mit nicht zulässigem Typ gelesen. Das Objekt-File
kann defekt sein.
```

Fehler: *empty input* (leeres Eingabefile)

```
Das Eingabe-File enthält keine Daten.
```

Fehler: *Read-Externals* (lese externe Symbole)

Externe Definitionen oder Referenzen unbekannten Typs wurden gelesen. Das Eingabe-File kann zerstört sein.

Fehler: *premature end of file* (frühzeitiges End-of-file)

Ein End-of-file (oder Out-of-data) wurde gefunden, obwohl weitere Daten erwartet wurden. Das Eingabe-File kann zerstört sein.

Fehler: *This utility can only be used on files that have NOT been passed through ALINK.* (Dieses Programm kann nur noch nicht mit ALINK bearbeitete Files verarbeiten.)

Das Eingabe-File wurde bereits von ALINK bearbeitet, enthält also keine externen Referenzen mehr, alle Module wurden bereits verkettet. Mit ATOM können jedoch nur Ausgabe-Files eines Assemblers oder C-Compilers verarbeitet werden.

8 ED – Der Bildschirm-Editor

Pfad: C:ED

Dieses Kapitel beschreibt den Bildschirm-Editor ED. Mit ihm können Sie Textfiles erstellen und verändern.

8.1 Einführung in den Bildschirm-Editor

Der Editor kann mit dem Befehl ED gestartet werden. Dabei kann entweder eine neue Datei angelegt oder eine vorhandene bearbeitet werden. Der Text wird auf dem Bildschirm angezeigt und kann vertikal oder horizontal gescrollt werden.

ED erkennt folgendes Muster an:

```
ED "FROM/A, SIZE/K"
```

Um ED aufzurufen, kann man zum Beispiel folgendes eingeben:

```
ED manfred
```

ED versucht, die mit »Manfred« bezeichnete Datei zu öffnen. »Manfred« ist damit die FROM-Datei. Kann sie geöffnet werden, liest ED die Datei in den Speicher und zeigt die ersten Zeilen auf dem Bildschirm an. Anderenfalls liefert ED ein leeres Blatt, bereit, einen neuen Text aufzunehmen. Um den Textspeicher zu vergrößern, so daß er größere Dateien aufnehmen kann, geben Sie den passenden Wert nach dem Schlüsselwort SIZE ein, zum Beispiel so:

```
ED manfred SIZE 45000
```

Die ursprüngliche Größe richtet sich nach der Größe der Datei, ist aber mindestens 40000 Byte groß.

Es kann übrigens nicht jede Dateiart mit ED eingelesen werden. So akzeptiert ED keine Quelldateien mit Binärcode. Solche Dateien können nur mit dem Zeilen-Editor EDIT überarbeitet werden.

Achtung! Endet eine Datei nicht mit einem Zeilenvorschub, fügt ED diesen an.

Während ED läuft, wird die unterste Zeile zur Ausgabe von Meldungen des Systems und als Befehlszeile benützt. Fehlermeldungen bleiben hier stehen, bis Sie einen neuen Befehl eingeben.

ED-Befehle werden in zwei Kategorien unterteilt:

- `unmittelbare Befehle`
- `erweiterte Befehle`

Man verwendet unmittelbare Befehle im Direkt-Modus; entsprechend werden erweiterte Befehle im erweiterten Modus verwendet. ED befindet sich zu Beginn des Editierens stets im Direkt-Modus. Um in den erweiterten Modus zu gelangen, betätigen Sie die [Esc]-Taste. Nachdem ED die danach eingegebenen Befehle ausgeführt hat, kehrt es automatisch in den Direkt-Modus zurück.

Im Direkt-Modus führt ED die Befehle sofort aus. Einen unmittelbaren Befehl geben Sie mit einer einzelnen Taste oder [Ctrl]-Tastenkombination ein. Für eine [Ctrl]-Tastenkombination halten Sie die [Ctrl]-Taste gedrückt, während Sie den dazugehörigen Buchstaben eingeben. So bedeutet [Ctrl]+[M], daß [Ctrl] gedrückt sein muß, während der Buchstabe »M« eingetippt wird.

Im erweiterten Modus erscheinen alle Eingaben in der Befehlszeile. ED führt keinen Befehl aus, solange Sie die Befehlszeile nicht beenden. Es können eine Reihe von erweiterten Befehlen in einer einzigen Befehlszeile stehen. Jeder Befehl kann ebenso mit anderen zu einer Gruppe zusammengefaßt werden. Die meisten unmittelbaren Befehle haben eine entsprechende erweiterte Version.

ED hält den Bildschirm stets auf dem aktuellen Stand. Geben Sie einen weiteren Befehl ein, während die Anzeige aktualisiert wird, führt ED den Befehl sofort aus, ändert aber die Anzeige bei nächster Gelegenheit. Die aktuelle Zeile wird immer zuerst angezeigt und ist auch immer auf dem neuesten Stand.

8.2 Unmittelbare Befehle

Dieser Abschnitt beschreibt die Befehle, die von ED sofort ausgeführt werden. Unmittelbare Befehle führen folgende Funktionen aus:

- `Cursorsteuerung`
- `Text einfügen`
- `Text löschen`
- `Text scrollen`
- `Befehle wiederholen`

8.2.1 Die Benutzung des Cursors

Um den Cursor um ein Zeichen in irgendeine Richtung zu bewegen, betätigen Sie die entsprechende Cursortaste rechts unten auf der Tastatur. Befindet sich der Cursor auf der rechten Seite des Bildschirms, scrollt ED den Text nach links, um den restlichen Text sichtbar zu machen. ED scrollt senkrecht eine Zeile auf einmal und waagerecht zehn Zeichen auf einmal. Der Cursor kann nicht außerhalb des Textes gebracht werden, das

heißt, er kann nicht vor den Textanfang, hinter das Textende oder über den linken Textrand hinaus gesetzt werden.

[Ctrl]+[]], das ist [Ctrl] und die schließende, eckige Klammer »]«, bringen den Cursor auf die rechte Seite der aktuellen Zeile. Nochmalige Eingabe von [Ctrl]+[]] bringt ihn wieder auf die linke Seite der Zeile zurück. Falls notwendig, wird der Text dazu waagerecht gescrollt. In gleicher Weise bringt [Ctrl]+[E] den Cursor an den Anfang der ersten Zeile des Bildschirms, falls er sich nicht schon dort befindet. Wenn der Cursor sich bereits dort befindet, bringt ihn [Ctrl]+[E] an das Ende der letzten Zeile des Bildschirms.

[Ctrl]+[T] bringt den Cursor an den Anfang des nächsten Wortes. [Ctrl]+[R] stellt den Cursor auf das Leerzeichen vor dem Wort. In beiden Fällen wird der Text, falls notwendig, senkrecht oder waagerecht gescrollt. Die Tab-Taste bringt den Cursor zur nächsten Tabulator-Position, dem Vielfachen der Tab-Einstellung 3. Es wird kein Tab-Zeichen in die Datei eingefügt.

8.2.2 Text einfügen

Jeder Buchstabe, den Sie im unmittelbaren Modus eintippen, erscheint an der aktuellen Cursorposition, falls die Zeile nicht zu lang ist (eine Zeile darf maximal aus 255 Zeichen bestehen). Versuchen Sie, mehr Zeichen einzugeben, nimmt ED keine weiteren an und gibt die folgende Meldung aus:

```
Line too long
```

In kürzeren Zeilen bewegt ED jedes Zeichen rechts vom Cursor weiter nach rechts, um neuem Text Platz zu machen. Wenn die Zeile den rechten Bildschirmrand erreicht, verschwindet die linke Seite der Zeile. ED zeigt wieder den Zeilenanfang, wenn der Text waagerecht gescrollt wird. Wird der Cursor über das Zeilenende hinaus bewegt, zum Beispiel mit [Tab] oder den [Ctrl]-Tasten, fügt ED Leerzeichen zwischen dem Zeilenende und jedem neu eingefügten Zeichen ein.

Um die aktuelle Zeile beim Cursor zu trennen und eine neue Zeile zu beginnen, betätigen Sie die [Return]-Taste. Ist der Cursor am Zeilenende, macht ED nach der aktuellen Zeile eine Leerzeile. Alternativ dazu können Sie mit [Ctrl]+[A] eine neue Leerzeile nach der gegenwärtigen erzeugen, wobei die aktuelle Zeile nicht aufgeteilt wird. In beiden Fällen erscheint der Cursor am linken Rand der neuen Zeile; voreingestellt ist Spalte 1.

Um sicherzustellen, daß ED automatisch das Zeichen für Wagenrücklauf erzeugt, können Sie einen rechten Rand setzen. Damit beendet ED die Zeile immer vor dem Wort, das über diesen Rand hinausgehen würde, und setzt dieses Wort und den Cursor in die nächste Zeile. Man nennt diesen Vorgang »Word-Wrap«. Beachten Sie bitte, daß diese automatische Randeinstellung nicht ordnungsgemäß arbeitet, wenn die Zeile keine Leerzeichen enthält, da ED nicht erkennen kann, wo der Zeilenumbruch stattfinden soll. Wird ein Zeichen eingegeben, und ist der Cursor am rechten Zeilenende, erzeugt ED automatisch eine neue Zeile. War das gerade eingegebene Zeichen kein Leerzeichen, setzt ED das Zeichen am Zeilenende in die neue Zeile. Wird Text innerhalb der Zeile eingefügt, sind also rechts vom Cursor bereits Zeichen vorhanden, arbeitet der automatische Zeilenumbruch nicht! Der

rechte Rand ist auf Spalte 79 voreingestellt. Der rechte Rand kann mit dem Befehl EX abgeschaltet werden. Weitere Einzelheiten zum Setzen des Randes finden Sie in Abschnitt 8.3.1 Programmsteuerung dieses Kapitels.

Haben Sie Text im falschen Schriftmodus geschrieben, zum Beispiel in Kleinschrift anstatt in Großschrift, kann dies mit Ctrl+F korrigiert werden. Bringen Sie den Cursor auf den ersten Buchstaben, der geändert werden soll, und geben Sie dann Ctrl+F ein. Wurde der Buchstabe in Kleinschrift geschrieben, ändert ihn Ctrl+F in einen Großbuchstaben. Wurde er in Großschrift geschrieben, ändert ihn Ctrl+F in einen Kleinbuchstaben. Befindet sich der Cursor auf keinem Buchstaben, zum Beispiel auf einem Leerzeichen oder einer Ziffer, ändert Ctrl+F nichts.

Ctrl+F setzt den Cursor um ein Zeichen nach rechts, auch wenn er auf ein Leerzeichen trifft. Ist jedoch das nächste Zeichen ein Buchstabe, kann die Groß-/Kleinschreibung mit Ctrl+F wiederum verändert werden. Der Befehl kann so lange wiederholt werden, bis alle Zeichen einer Zeile geändert sind. Geben Sie Ctrl+F nach dem letzten Buchstaben einer Textzeile wiederholt ein, wandert der Cursor bis zum Ende der Bildschirmzeile, auch wenn nichts verändert werden kann. Haben Sie diese Zeile:

```
Der Igel und der Hase gehen Hand in Hand
```

und Sie tippen über jedem Zeichen Ctrl+F, ändert sich die Zeile folgendermaßen:

```
dER iGEL UND DER hASE GEHEN hAND IN hAND
```

Aus der folgenden Zeile:

```
IF <datei> <= x
```

wird dann:

```
if <DATEI> <= X
```

Alle Buchstaben werden geändert, andere Zeichen und Ziffern aber beibehalten.

8.2.3 Löschen von Text

Die Backspace-Taste löscht das Zeichen links vom Cursor und bewegt dabei den Cursor um ein Zeichen nach links, wenn er sich nicht am Zeilenanfang befindet. Falls notwendig, scrollt ED dabei den Text. Die Del-Taste löscht das Zeichen unter dem Cursor, ohne daß dieser dabei bewegt wird. Wie bei jedem Löschen, werden Zeichen auf der Zeile nachgezogen und Text, der sich unsichtbar auf der rechten Seite befand, wird sichtbar. Ctrl+O dient zum Löschen von Worten. Die genaue Funktion hängt von dem Zeichen ab, auf dem sich der Cursor befindet. Ist dieses Zeichen ein Leerzeichen, löscht Ctrl+O alle Leerzeichen bis zum nächsten Zeichen auf der Zeile, das kein Leerzeichen ist. Anderenfalls löscht es alle Zeichen links der Cursorposition, bis ein Leerzeichen erscheint. Ctrl+Y löscht alle Zeichen von der Cursorposition bis zum Zeilenende.

Ctrl+B löscht die ganze aktuelle Zeile. Zum Löschen eines größeren Textblocks können Sie erweiterte Befehle einsetzen.

8.2.4 Scrollen des Textes

Mit den Tasten [Ctrl]+[U] und [Ctrl]+[D] wird der Text jeweils 12 Zeilen auf- oder abwärts gescrollt. [Ctrl]+[D] läßt den Cursor auf der Zeile, während der Text nach unten gescrollt wird.

[Ctrl]+[U] scrollt den Text hoch und bewegt den Cursor damit in der Datei 12 Zeilen weiter.

[Ctrl]+[V] bringt den ursprünglichen Bildschirminhalt zurück. Diese Funktion wird nur gebraucht, wenn ein weiteres Programm neben dem Editor den Bildschirm benutzt und ihn verändert. Im normalen Gebrauch erscheinen Meldungen anderer Programme im hinter dem Editorfenster liegenden Fenster.

8.2.5 Wiederholung von Anweisungen

Der Editor merkt sich jede eingegebene erweiterte Befehlszeile. Um diese Befehle jederzeit wieder auszuführen, brauchen Sie nur [Ctrl]+[G] einzugeben. Auf diese Weise kann man einen Suchbefehl als einen erweiterten Befehl aufstellen. Entspricht die erste gefundene Zeichenkette nicht der gewünschten, betätigen Sie [Ctrl]+[G], um die Suche fortzusetzen. Sie können komplexe Editierbefehle so viele Male ausführen, ohne sie neu eingeben zu müssen.

Geben Sie einen erweiterten Befehl als Befehlsgruppe mit einem Wiederholungszähler an, wiederholt ED die Befehle der Gruppe automatisch mehrmals. Sie brauchen dazu dann nicht [Ctrl]+[G] zu drücken. Im nächsten Abschnitt finden sich weitere Einzelheiten zu den erweiterten Befehlen.

8.3 Erweiterte Anweisungen

Dieser Abschnitt beschreibt die Befehle, die Ihnen im erweiterten Modus zur Verfügung stehen. Diese Befehle betreffen

- `Programmkontrolle`
- `Blockkontrolle`
- `Bewegung von Textteilen`
- `Text suchen`
- `Text ersetzen`
- `Text verändern`
- `Text einfügen`

In den erweiterten Modus gelangt man mit der Esc-Taste. Nachfolgende Eingaben erscheinen in der Befehlszeile am unteren Bildschirmrand. Fehler können Sie in der üblichen Weise mit der Backspace-Taste verbessern. Die ganze Zeile beenden Sie mit einem weiteren [Esc] oder mit der [Return]-Taste. Betätigen Sie [Esc], verbleibt der Editor im erweiterten Modus, nachdem die Befehlszeile ausgeführt wurde. Drücken Sie dagegen [Return], kehrt Amiga-DOS in den unmittelbaren Modus zurück. Um die Befehlszeile leer zu lassen,

geben Sie ein [Return] nach einem [Esc] ein. In diesem Fall kehrt ED in den unmittelbaren Modus zurück.

Erweiterte Befehle bestehen aus einem oder zwei Buchstaben, die in Groß- oder Kleinschrift eingegeben werden können. Sie können auch mehrere Befehle in der gleichen Befehlszeile eingeben, indem Sie sie durch ein Semikolon »;« trennen. Befehle können zusätzliche Argumente wie eine Zahl oder eine Zeichenkette besitzen. Eine Zeichenkette ist eine Folge von Buchstaben, die von einem Sonderzeichen eröffnet und beendet wird. Ein Sonderzeichen ist jedes Zeichen, das kein Buchstabe ist, zum Beispiel Leerzeichen, Semikolons oder Klammern. Gültige Zeichenketten könnten zum Beispiel so aussehen:

```
/Glück/!23 Meter!:Hallo!:"1/2"
```

Die meisten unmittelbaren Befehle haben eine entsprechende erweiterte Version. Eine vollständige Aufstellung findet sich in der Übersicht der erweiterten Befehle am Ende dieses Kapitels.

8.3.1 Programmsteuerung

Dieser Abschnitt beschreibt die Befehle zur Programmkontrolle. Diese Befehle sind: X (eXit), Q (Quit), SA (SAve), U (Undo), SH (SHow), ST (Set Tab), SL und SR (Set Left oder Set Right) und EX (EXtended).

Um die Arbeit mit ED zu beenden, geben Sie den Befehl X ein. Nach der Eingabe dieses Befehls schreibt ED den im Speicher befindlichen Text an ein Ausgabegerät oder in eine Zieldatei. Gibt man die Zieldatei aus, erkennt man, daß alle durchgeführten Änderungen vorhanden sind.

ED legt eine zeitweilige Sicherheitskopie der jeweils letzten Version einer geänderten Datei als »:T/ED-BACKUP« auf der Diskette ab. Diese Backup-Datei bleibt bestehen, bis man den Bildschirm-Editor verläßt. Dann überschreibt ED die Datei mit einem neuen Backup.

Wollen Sie den Editor verlassen, ohne daß Änderungen gespeichert werden, verwenden Sie den Befehl Q. Nach Q beendet ED die Arbeit unverzüglich, ohne eine Sicherheitskopie anzulegen und ohne die eingegebenen Änderungen abzuspeichern. Aus diesem Grund fragt ED bei geändertem Dateiinhalt nach, ob ein Ausstieg wirklich gewünscht wird.

Ein weiterer Befehl bewirkt, ähnlich einem Schnappschuß, eine Sicherung der Datei, wie sie gerade bearbeitet wird. Dies ist der Befehl SA. SA speichert den aktuellen Text in einer benannten Datei oder, falls kein Name angegeben, in der aktuellen Datei ab. Zum Beispiel:

```
SA !:Doc/gespeicherter.Text!
```

oder

```
SA
```

SA sollte immer zur Arbeitssicherung eingesetzt werden, wenn umfangreiche Veränderungen an einem großen Textfile gemacht werden sollen. Nichts ist ärgerlicher als der Verlust

vieler Stunden Arbeit, wenn gegen Ende der Strom ausfällt oder sich der Rechner aufhängt und alles löscht. Der Befehl SA gefolgt von Q entspricht übrigens dem Befehl X.

Werden irgendwelche Änderungen zwischen den Befehlen SA und Q eingegeben, erscheint folgende Meldung:

```
Edits will be lost type Y to confirm:
```

»Eingaben würden verlorengehen; geben Sie zur Bestätigung Y ein.« Haben Sie keine Änderungen gemacht, endet ED unmittelbar, der Inhalt der Quelldatei bleibt unverändert. Der Befehl SA ist auch deshalb nützlich, weil er die Eingabe eines anderen Dateinamens als des gegenwärtigen erlaubt. Aus diesem Grund ist es möglich, Kopien von verschiedenen Entwicklungsstadien eines Files zu machen und diese in verschiedenen Dateien oder Verzeichnisse abzulegen.

Um die jeweils letzte Änderung zurückzunehmen, wird der Befehl »U« verwendet. ED macht intern stets eine Kopie der Zeile, in der sich der Cursor gerade befindet, und ändert diese Kopie, wenn Zeichen hinzugefügt oder gelöscht werden. ED legt diese geänderte Kopie der Zeile erst in der Datei ab, wenn der Cursor aus der aktuellen Zeile bewegt wird. Dies gilt sowohl bei Gebrauch der Cursortasten; als auch beim Löschen oder Einfügen einer Zeile. Auch beim waagerechten und senkrechten Scrollen wird die geänderte Zeile übernommen. Der Befehl U löscht die geänderte Kopie und setzt die alte Version der Zeile ein.

Achtung! ED kann keine gelöschte Zeile zurückbringen. Sobald die aktuelle Zeile verlassen wurde, bleibt [Esc]-[U] wirkungslos. Der Befehl SH zeigt den aktuellen Status des Editors an. Auf dem Bildschirm werden Informationen ausgegeben, wie die Tabulatorstops, die Randeinstellungen, Blockmarkierungen, der Name der gerade editierten Datei und der belegte Pufferspeicher.

Zu Beginn jeder Sitzung setzt ED in jeder dritten Spalte einen Tabulator. Mit dem Befehl ST gefolgt von einer Zahl n ändern Sie die gegenwärtige Tabulatoreinstellung. ED setzt dann alle n Spalten einen Tabulator.

Den rechten und linken Rand des Textes legen Sie mit den Befehlen SR und SL gefolgt von einer Zahl, welche die gewünschte Spaltenposition bezeichnet, fest. Der linke Rand sollte dabei kleiner als die Bildschirmbreite sein.

Mit dem Befehl EX wird die Randeinstellung außer Kraft gesetzt. Geben Sie EX ein, dann nimmt ED in der aktuellen Zeile auf den rechten Rand keine Rücksicht mehr. Bewegt man den Cursor aus der aktuellen Zeile, aktiviert ED die Randeinstellung wieder.

8.3.2 Blocksteuerung

Soll ein Text bewegt, dupliziert oder gelöscht werden, verwendet man die in diesem Abschnitt erläuterten Blockkontroll-Befehle.

Ein Textblock wird mit den Befehlen BS (Blockanfang) und BE (Blockende) bezeichnet. Dazu bringen Sie den Cursor an eine beliebige Stelle der ersten Zeile des gewünschten Blockes, dort geben Sie den Befehl BS ein. Danach bewegen Sie den Cursor in die Zeile,

die die letzte Zeile des Blocks werden soll. Dort gibt man den Befehl BE zum Setzen des Blockendes ein. Haben Sie mit den Befehlen BS und BE einen Block festgelegt und ändern Sie den Text danach in irgendeiner Weise, sind Blockanfang und -ende nicht mehr definiert. Die einzige Ausnahme von dieser Regel bildet der Befehl IB (Insert Block) zum Einfügen des Textblocks.

Um nur eine Zeile als aktuellen Block zu bezeichnen, bringen Sie den Cursor auf die gewünschte Zeile und geben [Esc] und BS;BE ein. Die aktuelle Zeile wird dann zum aktuellen Block. Sie können einen Block nicht innerhalb einer Zeile beginnen oder beenden. (Der Block enthält immer ganze Zeilen.) Dazu müssen Sie zunächst die Zeile mit der Return-Taste aufteilen. Haben Sie einen Block definiert, können Sie eine Kopie davon mit dem Befehl IB (Insert Block) an beliebiger Stelle der Datei in den Text einfügen. Geben Sie diesen Befehl ein, fügt ED eine Kopie des Blocks unmittelbar hinter der aktuellen Zeile ein. Es können mehrere Kopien des Blocks eingefügt werden, solange der Block definiert ist, das heißt bis er verändert oder gelöscht wird.

Zum Löschen eines Blocks wird der Befehl DB (Delete Block) verwendet. DB löscht den mit den Befehlen BS und BE festgelegten Block. Sobald der Block gelöscht ist, gehen auch die Werte für Blockanfang und -ende verloren. Das bedeutet, daß ein Block nicht erst gelöscht und dann eine Kopie davon eingefügt werden kann. Die Befehlsfolge DB;IB bleibt also wirkungslos. Dagegen ist eine Kopie eines Blocks mit folgendem Löschen des Originals durchaus möglich (IB gefolgt von DB). Mit Blockmarkierungen können Sie Teile der Datei zur späteren Erinnerung markieren. Der Befehl SB (Show Block) zeigt immer die erste Zeile des Blocks in der ersten Bildschirmzeile. Um einen Block in eine andere Datei zu schreiben, wird der Befehl WB (Write Block) verwendet. Der Befehl benötigt ein Argument, das den Dateinamen bezeichnet. Der folgende Befehl:

```
WB !:Doc/Beispiel!
```

schreibt zum Beispiel den Blockinhalt in die Datei Beispiel, die sich im Verzeichnis Doc befindet. (Wird der Schrägstrich (/) zum Trennen von Verzeichnis und Dateien verwendet, sollte er nicht als Begrenzung für die Zeichenkette verwandt werden!) ED öffnet dazu eine neue Datei mit dem Namen, den Sie bestimmen. Dabei wird möglicherweise eine bestehende Datei mit gleichem Namen gelöscht und der Pufferinhalt hineingeschrieben! Mit dem Befehl IF (Insert File) fügen Sie eine Datei in die aktuelle Datei ein. ED liest dazu die Datei in den Speicher, die mit der nach IF eingegebenen Zeichenkette als Argument bezeichnet wurde. Die Datei wird unmittelbar nach der aktuellen Zeile eingefügt. Der folgende Befehl fügt die Datei :Doc/Beispiel unmittelbar nach der aktuellen Zeile in die gegenwärtige Datei ein:

```
IF !:Doc/Beispiel!
```

8.3.3 Verändern der Cursorposition

Der Befehl T bringt den Cursor an den Dateianfang, so daß sich die erste Zeile der Datei auf der ersten Bildschirmzeile befindet. Der Befehl B bewegt den Cursor an das Dateiende, so daß sich die letzte Zeile der Datei in der letzten Bildschirmzeile befindet.

Die Befehle N und P bringen den Cursor entsprechend an den Anfang der nächsten Zeile (N) und der vorhergehenden Zeile (P). Die Befehle CL und CR bewegen den Cursor jeweils ein Zeichen nach links (CL) oder rechts (CR), während CS ihn an den Anfang und CE ans Ende der Zeile bringt. Der Befehl M bewegt den Cursor zu einer bestimmten Zeile. Nach M wird die Zeilennummer eingegeben, die die aktuelle Zeile werden soll. Der folgende Befehl bringt den Cursor zum Beispiel in Zeile 503 der Datei:

```
M 503
```

Der Befehl M ist eine schnelle Art, um an eine bekannte Position in der Datei zu gelangen. Natürlich kann auch zum Befehl N eine Wiederholungszahl angegeben werden, um in die richtige Zeile der Datei zu gelangen, dies ist aber bedeutend langsamer, als die eben beschriebene Methode.

8.3.4 Suchen und Ersetzen von Text

Das Bildschirmfenster kann aber auch mit dem Befehl F (Find) zu einem einzelnen Begriff bewegt werden; dem Befehl F folgt eine Zeichenkette, die den Text bezeichnet, der angesprungen werden soll. Die Suche beginnt ein Zeichen vor der aktuellen Cursorposition und wird bis zum Dateiende fortgesetzt. Wird die Zeichenkette gefunden, so erscheint der Cursor am Anfang der gefundenen Zeichenkette. Die Zeile wird zur ersten Zeile auf dem Bildschirm.

Um den Text rückwärts zu durchsuchen, verwenden Sie den Befehl BF (Backward Find) auf die gleiche Weise wie F. BF findet die letzte vorhandene Zeichenkette vor der aktuellen Cursorposition. BF sucht links vom Cursor und dann alle Zeilen rückwärts bis zum Dateianfang nach der Zeichenkette. Um die erste Stelle der Datei zu finden, in der die Zeichenkette auftritt, verwenden Sie T (Top-of-file) gefolgt von F. Die letzte Stelle findet man mit B (Bottom-of-file) gefolgt von BF.

Der Befehl E tauscht zwei durch Abgrenzer getrennte Zeichenketten gegeneinander aus, die erste Zeichenkette, die bereits im Text vorhanden ist, wird durch die zweite ersetzt. So ersetzt zum Beispiel

```
E/Anton/Fritz/
```

das Wort Anton an der ersten Stelle, an der es auftritt, durch Fritz. Der Editor beginnt die Suche nach der ersten Zeichenkette an der aktuellen Cursorposition und fährt damit bis zum Dateiende fort. Sobald der Austausch erledigt ist, springt der Cursor zum Ende des ersetzten Textes.

Leere Zeichenketten bezeichnen Sie durch Eingabe von zwei Abgrenzern ohne Zwischenraum. Ist die erste, die Such-Zeichenkette im obigen Beispiel, leer, fügt der Editor die zweite Zeichenkette an der aktuellen Cursorposition ein. Ist die zweite Zeichenkette leer, bewirkt das ein Löschen der gesuchten Zeichenkette. Der folgende Befehl löscht das nächste Auftreten von Anton im Text:

```
E/Anton//
```

Während des Austauschens von Textteilen ignoriert ED übrigens alle Randbegrenzungen.

Der Befehl EQ (Exchange and Query) ist eine Variante des E-Befehls. Geben Sie den Befehl EQ ein, fragt ED Sie, ob der Austausch stattfinden soll. Dies ist sinnvoll, wenn nur in einigen Fällen ersetzt werden soll und in anderen nicht. So erscheint nach Eingabe von

```
EQ /Anton/Fritz/
```

folgende Meldung:

```
Exchange?
```

in der Befehlszeile. Wird nun ein »N« getippt, geht der Cursor an der gesuchten Zeichenkette vorüber. Tippen Sie hingegen »Y«, wird wie gewohnt ausgetauscht.

Die Suche-und-ersetze-Befehle machen gewöhnlich während der Suche einen Unterschied zwischen Groß- und Kleinschreibung. Um Groß- und Kleinschreibung zu ignorieren, brauchen Sie nur den Befehl UC zu erteilen. Nach Gebrauch von UC paßt zu der Such-Zeichenkette »Anton« auch »ANTON«, »AntoN« oder »AnTON«. Soll ED wieder zwischen Groß- und Kleinschreibung unterscheiden, können Sie UC durch LC wieder aufheben.

8.3.5 Text ändern

Sie können den Befehl E nicht zum Einfügen einer neuen Zeile in den bestehenden Text verwenden. Statt dessen gibt es dafür die Kommandos I und A. Geben Sie nach dem Befehl I (Insert before) eine Zeichenkette ein, die in den Text eingefügt werden soll. ED fügt diese dann vor der aktuellen Zeile ein. Zum Beispiel:

```
I /Füge dies VOR der aktuellen Zeile ein/
```

setzt »Füge dies VOR der aktuellen Zeile ein« als eine neue, eigenständige Zeile vor die aktuelle Zeile, in der der Cursor steht. Den Befehl A (insert After) verwenden Sie in der gleichen Weise, nur wird die neue Zeile nach der aktuellen eingefügt. So setzt

```
A /Füge dies NACH der aktuellen Zeile ein/
```

die Zeile »Füge dies NACH der aktuellen Zeile ein« als eine neue Zeile nach der aktuellen ein, in der der Cursor steht. Mit dem Befehl S trennen Sie die aktuelle Zeile an der Cursorposition. Im erweiterten Modus hat S die Funktion der Return-Taste im Direktmodus. In Abschnitt 8.2.2 finden sich weitere Details zum Trennen von Zeilen.

Der Befehl J fügt die nächste Zeile an das Ende der aktuellen an.

Der Befehl D löscht die aktuelle Zeile in der gleicher Weise wie [Ctrl]+[B] im unmittelbaren Modus. Der Befehl DC löscht das Zeichen auf der Cursorposition ebenso wie die [Del]-Taste.

8.3.6 Wiederholte Befehle

Um einen Befehl mehrmals zu wiederholen, stellen Sie ihm die Anzahl der Wiederholungen voran. Der Befehl

```
4 E /gehen/wandern
```

ändert die nächsten vier Stellen, an denen »gehen« vorkommt, in »wandern« um. ED bringt den Bildschirm nach jeder Änderung auf den neuesten Stand.

Mit dem Befehl RP (RePeat) wird ein Befehl ausgeführt, bis ein Fehler auftritt, zum Beispiel weil das Dateiende erreicht ist.

```
T; RP E /gehen/wandern
```

ändert alle Stellen, an denen »gehen« vorkommt, in »wandern« um.

Das Kommando T wird vorangestellt, um sicherzugehen, daß alle vorkommenden Worte »gehen« geändert werden. Wird T weggelassen, werden Änderungen nur ab der aktuellen Cursorposition ausgeführt.

Befehlssequenzen werden wiederholt ausgeführt, wenn sie in runden Klammern zusammengefaßt wurden. Sie können Befehlssequenzen übrigens auch ineinander verschachteln. Der Befehl

```
RP (F /Hilfestellung/; 3 A//)
```

fügt zum Beispiel drei Leerzeilen (Kopien des Leer-Strings) nach jeder Zeile ein, die das Wort »Hilfestellung« enthält. Beachten Sie bitte, daß diese Befehlszeile nur die Zeilen ab der Cursorposition bis zum Dateiende abarbeitet. Damit die Befehlsfolge die ganze Datei erfaßt, muß vorher an den Dateianfang gesprungen werden.

Einige Befehle sind zwar möglich, aber sinnlos. Zum Beispiel:

```
RP SR 60
```

setzt den Rand ab Spalte 60 und wiederholt diesen Befehl bis in alle Ewigkeit. Jede Folge an erweiterten Befehlen, auch solche, die wiederholt abgearbeitet werden, können Sie jedoch durch Eingabe eines beliebigen Zeichens abbrechen.

8.3.7 Kurzübersicht der ED-Befehle

Sondertasten

Befehl	Funktion
[Backspace]	Löscht das Zeichen links vom Cursor.
[Del]	Löscht das Zeichen an der Cursorposition.
[Esc]	Einschalten des erweiterten Modus.
[Return]	Trennt Zeile an Cursorposition und erzeugt eine neue Zeile.
[Tab]	Setzt Cursor nach rechts an die nächste Tabulatorposition. (Fügt kein Tab-Zeichen ein!)
< >	Bewegt Cursor nach oben.
< >	Bewegt Cursor nach unten.
< >	Bewegt Cursor nach rechts.
< >	Bewegt Cursor nach links.

Unmittelbare Befehle

Befehl	Funktion
Ctrl+A	Fügt eine Zeile ein.
Ctrl+B	Löscht eine Zeile.
Ctrl+D	Scrollt Text nach unten.
Ctrl+E	Setzt Cursor an Bildschirmanfang oder -ende.
Ctrl+F	Ändert Groß-/Kleinschreibung.
Ctrl+G	Wiederholt die letzte erweiterte Befehlszeile.
Ctrl+H	Löscht das Zeichen links vom Cursor (wie Backspace).
Ctrl+I	Bewegt Cursor zur nächsten rechten Tabulatorposition.
Ctrl+M	Return.
Ctrl+O	Löscht Wort oder Leerzeichen.
Ctrl+R	Cursor an das Ende des vorhergehenden Wortes.
Ctrl+T	Cursor an den Anfang des nächsten Wortes.
Ctrl+U	Scrollt Text nach oben.
Ctrl+V	Bringt den ursprünglichen Bildschirminhalt zurück.
Ctrl+Y	Löscht bis zum Ende der Zeile.
Ctrl+[	Einschalten des erweiterten Modus (wie Esc).
Ctrl+]	Cursor an Zeilenanfang oder -ende setzen.

Erweiterte Befehle

Es folgt eine vollständige Aufstellung der erweiterten Befehle, einschließlich derer, die nur eine erweiterte Version von unmittelbaren sind. In der Aufstellung bedeuten die drei Zeichen /s/, daß dem Befehl eine Zeichenkette folgt. Die Zeichen /s/t/ bedeuten, daß zwei Zeichenketten ausgetauscht werden sollen. »n« steht für eine Zahl.

Befehl	Funktion
A/s/	Einfügen einer Zeile hinter der aktuellen Zeile.
B	Cursor zum Dateiende.
BE	Cursorposition ist Blockende.
BF/s/	Sucht Zeichenkette nach oben.
BS	Cursorposition ist Blockanfang.
CE	Cursor an das Zeilenende setzen.
CL	Cursor um ein Zeichen nach links setzen.
CR	Cursor um ein Zeichen nach rechts setzen.
CS	Cursor an den Zeilenanfang setzen.
D	Löscht die aktuelle Zeile.
DB	Löscht Block.
DC	Löscht Zeichen an der Cursorposition.
E /s/t/	Ersetzt s durch t.
EQ/s/t/	Ersetzt wie E, aber erst nach Rückfrage.

Befehl	Funktion
EX	Ignoriert den rechten Rand.
F /s/	Sucht Zeichenkette s.
I /s/	Fügt Zeichenkette vor der aktuellen Zeile ein.
IB	Fügt Kopie eines Blocks vor der aktuellen Zeile ein.
IF /s/	Fügt Datei vor der aktuellen Zeile ein.
J	Verbindet die aktuelle Zeile mit der nächsten.
LC	Beim Suchen zwischen Groß- und Kleinschreibung unterscheiden.
M n	Bewegt den Cursor zur Zeile Nummer n.
N	Bewegt den Cursor zur nächsten Zeile.
P	Bewegt den Cursor zur vorhergehenden Zeile.
Q	ED wird ohne Speicherung des Texts verlassen.
RP	Wiederholt bis zu einem Fehler.
S	Trennt Zeile an Cursorposition.
SA /s/	Speichert Text in eine Datei s.
SB	Zeigt Block auf dem Bildschirm.
SH	Zeigt Informationen über den Status des Editors an.
SL n	Setzt linken Rand an Position n.
SR n	Setzt rechten Rand an Position n.
ST n	Setzt Tabulatorabstand n fest.
T	Cursor zum Dateianfang setzen.
U	Nimmt letzte Eingabe in der aktuellen Zeile zurück.
UC	Beim Suchen Groß- und Kleinschreibung ignorieren.
WB/s/	Schreibt Block in eine Datei s.
X	Verlassen des Programmes ED mit vorherigem Speichern des Textes.

9 EDIT – Der Zeileneditor

Pfad: C:EDIT

Dieses Kapitel beschreibt im einzelnen die Handhabung des Zeileneditor EDIT. Der erste Teil führt den Leser in den Editor ein. Der zweite Teil beschreibt alle Editorfunktionen vollständig. Eine Kurzübersicht über alle EDIT-Befehle findet sich am Ende des Kapitels.

9.1 EDIT – Ein zeilenorientierter Texteditor

EDIT ist ein Texteditor, der sequentielle Dateien Zeile für Zeile abarbeitet. EDIT bewegt sich durch die Eingabe- oder Quelldatei, wobei jede Zeile, nach eventuellen Änderungen, in eine sequentielle Ausgabedatei, die Zieldatei, übernommen wird. Der Ablauf von EDIT erzeugt eine Kopie der Quelldatei, die alle Änderungen enthält, die Sie mit den Editier-Befehlen gefordert haben.

Obwohl EDIT die Ursprungsdatei, auch Quelldatei genannt, üblicherweise Wort für Wort, also sequentiell abarbeitet, läßt sich auch eine begrenzte Zahl von zurückliegenden Zeilen bearbeiten. Dies ist möglich, da EDIT die abgearbeiteten Zeilen nicht sofort in die Zieldatei schreibt, sondern sie statt dessen in einem Ausgabe-Puffer ablegt. Die Größe dieses Puffers hängt vom verfügbaren Speicherplatz ab. Sollen sehr viele Informationen im Speicher abgelegt werden, können Sie mit der EDIT-Option OPT, die im nächsten Abschnitt beschrieben wird, den Speicher vergrößern. Jeder Text kann mehrmals abgearbeitet werden.

Mit den EDIT-Befehlen können Sie:

- Teile der Quelldatei ändern.
- Teile der Quelldatei an andere Dateien ausgeben.
- Material anderer Dateien einfügen.

9.1.1 Aufrufen von EDIT

Dieser Abschnitt beschreibt die Syntax von EDIT und die Parameter, die beim Aufruf an EDIT übergeben werden können oder müssen. Edit wird mit folgendem Muster aufgerufen:

```
EDIT "FROM/A,TO,WITH/K,VER/K,OPT/K"
```

Das Muster eines Befehls ist in Kapitel 1 dieses Handbuches genauer beschrieben. Amiga-DOS erlaubt alle in diesem Muster angegebenen Argumente in der angegebenen Form. Einige der Argumente sind optional. Das heißt, sie werden nur bei Bedarf eingegeben. Einige Argumente ohne »/A« werden von Amiga-DOS alleine durch ihre Position richtig zugeordnet. Argumente, denen in der Musterbeschreibung ein »/A« angehängt ist (wie zum Beispiel FROM), müssen beim Aufruf von EDIT angegeben werden. Das zugehörige Schlüsselwort aber kann entfallen. Argumente, denen ein »/K« folgt (zum Beispiel WITH), sind optional, werden sie aber angegeben, muß auch das Schlüsselwort mit erscheinen. Einige müssen mit dem zugehörigen Schlüsselwort erscheinen. Sollten Sie die Syntax von EDIT nicht mehr wissen, genügt es, den folgenden Befehl einzutippen:

```
EDIT ?
```

Amiga-DOS zeigt dann das vollständige Muster des Befehls. Das Format von EDIT lautet:

```
EDIT [FROM]<file>[[TO]<file>][WITH <file>][VER<file>]
     [OPT Pn|Wn|PnWn]
```

Nach dem Start von EDIT wird übrigens ein Verzeichnis namens »:T« eingerichtet, falls es noch nicht existiert. Dieses Verzeichnis dient der Aufnahme temporärer Dateien. Mehr Informationen über diese Muster und die Syntax von Befehlen finden Sie in den Kapiteln 2 und 3 dieses Handbuchs. Das Argument FROM deklariert die Quelldatei, die editiert werden soll. Das Argument muß auftreten, das Schlüsselwort selbst kann, muß aber nicht, angegeben werden; das heißt, Amiga-DOS erkennt die FROM-Datei an ihrer Position. Das TO steht für die Zieldatei. Das ist die Datei, in der EDIT die Ausgabe mit allen Änderungen ablegen soll. Wird das TO-Argument weggelassen, benutzt EDIT eine interne Datei, die unter dem Namen der FROM-Datei abgespeichert wird, wenn der Editiervorgang beendet ist. Wird der EDIT-Befehl STOP eingesetzt, findet diese Umbenennung nicht statt und die FROM-Datei bleibt unverändert erhalten.

Das Schlüsselwort WITH steht für die Datei, welche die Editierbefehle enthält. Lassen Sie das Argument WITH fort, liest EDIT Befehle von der Tastatur.

Das Schlüsselwort VER steht für die Datei, in die EDIT Fehlermeldungen und Zeilenbestätigungen schreibt. Wird VER weggelassen, verwendet EDIT auch hier das Terminal. Mit dem Schlüsselwort OPT schließlich werden die möglichen Optionen angegeben. Gültige Optionen sind P(n), womit die Zeilenzahl auf die ganzzahlige Variable (n) eingestellt wird, und W(n), sie legt die maximale Zeilenlänge auf (n) Zeichen fest. Falls Sie die Werte nicht ändern, sind sie auf P40, W120 eingestellt. OPT können Sie indirekt zum Vergrößern oder Verkleinern des verfügbaren Speichers verwenden. EDIT verwendet P*W, also die Zahl der maximalen Zeilen multipliziert mit der Zeilenbreite, um den benötigten Speicherplatz festzulegen. Die Speichergröße verändern Sie durch Einstellen der P und W zugeordneten Zahlen. P50 stellt mehr Speicher als üblich zur Verfügung, P30 weniger.

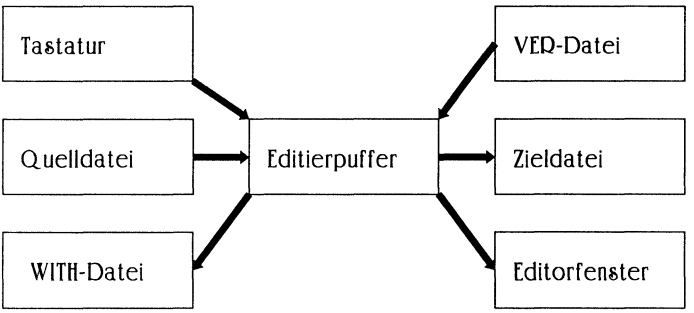


Bild 9.1: Die Funktionsweise von EDIT

Hier nun einige Muster-Befehle zum Aufruf von EDIT:

```
EDIT prog1 TO prog1neu WITH editierbfehlen
EDIT prog1 OPT P50W240
EDIT prog1 VER verdatei
```

Achtung! Im Gegensatz zu ED können Sie mit EDIT keine neue Datei anlegen. Versuchen Sie dies, gibt Amiga-DOS eine Fehlermeldung aus, weil es die neue Datei im aktuellen Verzeichnis nicht finden kann.

9.1.2 EDIT-Kommandos einsetzen

Dieser Abschnitt führt einige der grundlegenden EDIT-Befehle ein, wobei viele der weiterführenden Möglichkeiten weggelassen werden. Eine vollständige Beschreibung der Befehlssyntax erfolgt in Abschnitt 9.2 dieses Kapitels.

9.1.2.1 Die aktuelle Zeile

Wenn EDIT von der Quelldatei Zeilen einliest und sie dann in die Zieldatei schreibt, wird die Zeile, die gerade bearbeitet wird, als die aktuelle Zeile bezeichnet. EDIT führt alle Textänderungen in der aktuellen Zeile aus. EDIT fügt neue Zeilen stets vor die aktuelle Zeile ein. Wird EDIT gestartet, wird die erste Zeile der Quelldatei die aktuelle Zeile.

9.1.2.2 Zeilennummern

EDIT verbindet jede Zeile der Quelldatei mit einer Zeilennummer. Diese Zeilennummer ist kein Teil der in der Datei gespeicherten Information, sondern EDIT weist sie durch Zählen der eingelesenen Zeilen zu. Bei der Arbeit mit EDIT können Sie auf die Zeilennummer verweisen. Eine eingelesene Zeile behält, solange sie sich im Hauptspeicher befindet, ihre Zeilennummer bei, auch wenn Zeilen vor oder nach ihr gelöscht oder eingefügt werden. Die Zeilennummern bleiben unverändert, bis Sie die Arbeit beenden oder die Zeilen mit

dem Befehl »=« umnumerieren. Immer wenn Sie eine Datei aufrufen, weist EDIT Zeilennummern zu. Beim wiederholten Aufruf der Datei unterscheiden sich die Zeilennummern aber unter Umständen.

9.1.2.3 Auswahl der aktuellen Zeile

Eine aktuelle Zeile können Sie auf drei Arten ansprechen:

- durch Abzählen der Zeilen
- durch Angeben des Inhaltes
- durch Angeben einer Zeilennummer

Diese drei Methoden werden nun beschrieben:

Abzählen der Zeilen:

Die Befehle N und P erlauben den Sprung zur nächsten oder vorhergehenden Zeile. Geben Sie vor diesen Befehlen eine Nummer an, können Sie diese Anzahl an Zeilen vorwärts oder rückwärts springen. Um eine Zeile vorwärts zu springen, müssen Sie zum

Beispiel eingeben:

```
N
```

Jeden EDIT-Befehl können Sie sowohl in Großschrift als auch in Kleinschrift eingeben.

Um vier Zeilen vorwärts zu springen, geben Sie ein:

```
4 N
```

Die vierte Zeile nach der aktuellen wird zur neuen aktuellen Zeile.

Um eine Zeile zurückzuspringen, tippen Sie:

```
P
```

Der Befehl P erlaubt ebenfalls die Angabe von Zahlen. Geben Sie zum Beispiel den folgenden Befehl ein, wird die vierte Zeile vor der aktuellen Zeile zur neuen aktuellen Zeile:

```
4 P
```

Sie können nur zu Zeilen zurückspringen, die EDIT noch nicht in die Ausgabedatei geschrieben hat. EDIT erlaubt normalerweise nur ein Zurückspringen um 40 Zeilen. Um weiter zurückspringen zu können, müssen Sie beim Aufruf von EDIT mit der Option P mehr Zeilen einrichten. Dazu finden sich weitere Einzelheiten im Abschnitt 9.1.1 dieses Kapitels.

Eine spezielle Zeilennummer anspringen:

Der Befehl M erlaubt, durch Angabe der Zeilennummer eine neue aktuelle Zeile anzuwählen. Dazu werden der Befehl M und die gewünschte Zeilennummer eingegeben. So weist zum Beispiel der Befehl M 45 (Move 45) EDIT an, zu Zeile 45 zu springen. Befindet man sich hinter der Zeile 45, so wird auch zu ihr zurückgesprungen, vorausgesetzt, daß sie sich noch im Speicher befindet.

Die Angabe einer Zeilennummer und das Abzählen der Zeilen können miteinander kombiniert werden. Zum Beispiel:

```
M12; 3N
```

springt zu Zeile15.

Aufeinanderfolgende Befehle in der gleichen Zeile trennen Sie durch ein Semikolon (;).

Auswahl einer Zeile durch ihren Inhalt:

Mit dem Befehl F (Find) können Sie eine aktuelle Zeile durch ihren Inhalt aufsuchen lassen. Zum Beispiel:

```
F/Inhalt/
```

findet die nächste Zeile, in der sich der Begriff »Inhalt« befindet. Die Suche beginnt bei der aktuellen Zeile und geht vorwärts durch die gesamte Quelldatei, bis die benötigte Zeile gefunden ist. Falls EDIT das Dateiende erreicht hat, ohne eine passende Zeile zu finden, wird folgende Meldung ausgegeben:

```
INPUT EXHAUSTED
```

Ebenso ist es möglich, mit dem Befehl BF (Backwards Find) rückwärts zu suchen:

```
BF/Zucker und Zimt/
```

Die Suche beginnt in der aktuellen Zeile, EDIT bewegt sich nun aber rückwärts in der Datei, bis die gewünschte Zeile gefunden ist. Erreicht EDIT den Anfang der Datei, ohne daß eine passende Zeile gefunden wird, erscheint folgende Meldung:

```
NO MORE PREVIOUS LINES
```

Beachten Sie bitte, daß in den obigen Beispielen der gesuchte Text zwischen einzelnen Schrägstrichen (/) steht. Dieser angegebene Text wird als Zeichenkette bezeichnet. Die Zeichen, die Sie zum Anzeigen des Anfangs und des Endes einer Zeichenkette verwenden, werden Abgrenzungszeichen genannt. Einige spezielle Zeichen, wie : . , und * sind als Abgrenzer verwendbar; natürlich darf die Zeichenkette selbst keine Abgrenzer enthalten. EDIT ignoriert die Leerzeichen zwischen Befehl und dem ersten Abgrenzer, beachtet aber Leerzeichen innerhalb der Zeichenkette, der Inhalt wird genauestens verglichen. Zum Beispiel:

```
F /Da-Da Baum/
```

findet »DA-DA Baum« oder »Da-Da Baum« nicht.

F ohne Argument wiederholt die vorhergehende Suchoperation. Der folgende Befehl zum Beispiel findet die zweite Zeile im File, in der »grauer Vogel« steht.

```
F/grauer Vogel/; N; F
```

Der Befehl N zwischen den beiden F-Befehlen ist notwendig, da F immer in der aktuellen Zeile mit der Suche beginnt. Lassen Sie N fort, findet das zweite F die gleiche Stelle in der aktuellen Zeile wie das erste.

9.1.2.4 Qualifikatoren

Die oben beschriebene, grundlegende Form des Befehls F findet eine Zeile, die die angegebene Zeichenkette an irgendeiner Stelle enthält. Um die Suche auf den Anfang oder das Ende einer Zeile zu beschränken, stellen Sie dem Suchbegriff den Buchstaben B (für Beginn) oder E (für Ende) voran. In diesem Fall müssen Sie nach dem F ein oder mehrere Leerzeichen eingeben.

Der folgende Befehl:

```
F B /glückliche Zeiten/
```

findet eine Zeile, die mit »glückliche Zeiten« beginnt, während

```
F E /Hosenträger/
```

eine Zeile findet, die mit »Hosenträger« endet. Werden dem gesuchten Inhalt genauere Bezeichnungen angehängt, beschleunigt sich die Suche, da EDIT nur einen Teil jeder Zeile durchsuchen muß.

In der oben angegebenen Form sind B und E Beispiele für Qualifikatoren und das ganze Argument wird als qualifizierte Zeichenkette bezeichnet. Weitere Qualifikatoren stehen zur Verfügung. Zum Beispiel der Qualifikator P:

```
F P /Unter den Linden/
```

bedeutet, daß eine Zeile gesucht wird, die ausschließlich den Text »Unter den Linden« enthält. Die gewünschte Zeile darf keine weiteren Zeichen vor oder nach der angegebenen Zeichenkette enthalten. Das heißt, nach Eingabe dieses Befehls findet EDIT die Zeile:

```
Unter den Linden
```

Die folgende Zeile jedoch wird nicht gefunden:

```
Unter den Linden.
```

Um eine leere Zeile zu finden, können Sie eine leere Zeichenkette mit dem Qualifikator P verwenden. Zum Beispiel:

```
F P//
```

Sie können mehrere Qualifikatoren in beliebiger Reihenfolge gebrauchen.

9.1.2.5 Ändern der aktuellen Zeile

Dieser Abschnitt beschreibt den Einsatz der Befehle E, A und B zum Ändern des Textes in der aktuellen Zeile. Der folgende Befehl zum Beispiel:

```
E/Wunderland/Fernglas/
```

ersetzt die Zeichenkette »Wunderland« durch den Begriff »Fernglas«. Beachten Sie bitte, daß Sie in der Mitte nur einen einzelnen Abgrenzer zum Trennen der beiden Zeichenketten benutzen. Um Teile des Textes zu löschen, das heißt, Text durch nichts zu ersetzen, geben Sie eine leere zweite Zeichenkette wie folgt an:

```
E/Gitarre//
```

Um einer Zeile neuen Text anzufügen, verwenden Sie die Befehle A oder B. Das Kommando A fügt die zweite Zeichenkette nach der ersten ein. In gleicher Weise fügt der Befehl B die zweite Zeichenkette vor dem ersten Erscheinen der ersten ein. Wenn zum Beispiel die aktuelle Zeile den Text

```
acht ist gleich acht
```

enthält und die folgende Befehlsfolge eingegeben wird:

```
A/acht/zig/; B L /acht/zweiundsiebzig und /
```

ändert sich der Text in den folgenden:

```
achtzig ist gleich zweiundsiebzig und acht
```

Würden Sie den Qualifikator L bei dem Befehl B weglassen, sähe das Ergebnis so aus:

```
zweiundsiebzig und achtzig ist gleich acht
```

da von links nach rechts gesucht wird und EDIT bei der ersten gefundenen Stelle beginnt. Der Qualifikator L bedeutet, daß der Befehl am Zeilenende mit der Suche beginnen soll.

Ist das erste Argument in einem A-, B- oder E-Befehl leer, fügt EDIT die zweite Zeichenkette an den Anfang oder das Ende einer Zeile ein. Um die Position der zweiten Zeichenkette näher zu bezeichnen, verwenden Sie die Qualifikatoren L oder E oder lassen sie weg.

Geben Sie einen A-, B- oder E-Befehl ein, der den Suchbegriff nicht findet, erscheint folgende Meldung entweder auf dem Bildschirm oder in der VER-Datei, sofern Sie eine beim Aufrufen von EDIT angegeben haben:

```
NO MATCH
```

9.1.2.6 Löschen ganzer Zeilen

Dieser Abschnitt beschreibt, wie Zeilen aus der Datei entfernt werden können. Um eine Reihe von Zeilen zu löschen, können Sie ihre Zeilennummern in einem D-Befehl verwenden. Sie wenden den Befehl D an, indem Sie D und die Zeilennummer eingeben. Geben Sie nach der Zeilennummer ein Leerzeichen und eine zweite Zahl ein, löscht EDIT alle Zeilen von der ersten Zeilennummer bis zur letzten. Zum Beispiel:

```
D97 104
```

löscht die Zeilen 97 bis 104 einschließlich, Zeile 105 ist dann die neue aktuelle Zeile. Die aktuelle Zeile löschen Sie durch Eingabe von D ohne weitere Angaben. Der folgende Befehl:

```
F /Erdbeereis/; D
```

löscht zum Beispiel die Zeile, die »Erdbeereis« enthält, und die darauffolgende Zeile wird zur aktuellen. Sie können eine qualifizierte Suche mit dem Befehl »D« wie folgt kombinieren:

```
F B/Die/; 4D
```

Diese Befehlsfolge löscht vier nacheinander folgende Zeilen, deren erste Zeile mit »Die« beginnt.

Statt einer Zeilennummer können Sie auch einen Punkt (.) oder einen Stern (*) eingeben. Mit dem Punkt verweisen Sie auf die aktuelle Zeile. Auf das Dateiende verweisen Sie mit dem Stern. Der folgende Befehl löscht zum Beispiel den restlichen Text der Quelldatei einschließlich der aktuellen Zeile:

```
D . *
```

9.1.2.7 Einfügen neuer Zeilen

Dieser Abschnitt beschreibt, wie mit Hilfe von EDIT neue Zeilen in die Datei eingefügt werden. Um eine oder mehrere Zeilen vor einer angegebenen Zeile einzufügen, verwenden Sie den Befehl I (Insert). Sie können den Befehl I einzeln oder in Verbindung mit einer Zeilennummer, einem Punkt (.) oder einem Stern (*) eingeben. EDIT fügt den Text vor der aktuellen Zeile ein, wenn Sie nur I oder zusammen mit einem Punkt (.) angeben. Geben Sie nach I einen Stern (*) ein, wird der Text am Dateiende angefügt; das heißt nach der letzten Zeile der Datei. Jeder neu eingegebene Text wird vor der angegebenen Zeile eingefügt. Das Ende des eingefügten Textes zeigen Sie an, indem Sie [Return] betätigen, [Z] eingeben und nochmals [Return] drücken. Zum Beispiel:

```
I 468
Die kleinen Fische im Meer,
antworten Dir nimmer mehr.
Z
```

fügt die beiden Textzeilen vor Zeile 468 ein.

Lassen Sie die Zeilennummer nach dem Befehl fort, fügt EDIT den neuen Text vor der aktuellen Zeile ein. Zum Beispiel:

```
F /Wecker/; I
Er sagte: "Ich werde ihn aufwecken gehen, wenn..."
Z
```

Dieser Mehrfach-Befehl sucht die Zeile, die den Text »Wecker« enthält und damit auch zur aktuellen Zeile wird, und fügt die dahinter angegebene neue Zeile ein.

Nach dem I-Befehl mit einer Zeilennummer ist die aktuelle Zeile stets die Zeile mit der angegebenen Nummer; anderenfalls bleibt die aktuelle Zeile unverändert.

Um langes Eintippen zu ersparen, bietet EDIT den Befehl R (Replace), der genau der Eingabe von DI, also D für Löschen und I für Einfügen, entspricht. Zum Beispiel:

```
R 19 26
Wenn im Winter die Felder weiß sind
Z
```

löscht die Zeilen 19 bis einschließlich 26 und fügt dann den neuen Text vor Zeile 27, die nun die aktuelle Zeile geworden ist, ein.

9.1.2.8 Wiederholen von Befehlen

Viele EDIT-Befehle können automatisch wiederholt werden, wie oben in den Beispielen zu N und D gezeigt wurde. Zudem kann eine Gruppe von Befehlen wiederholt werden, indem sie in Klammern gesetzt zu einer Befehlsgruppe formiert wird. Der folgende Befehl löscht die nächsten sechs leeren Zeilen in der Quelldatei:

```
6 (F P//; D)
```

Befehlsgruppen dürfen nur eine Zeile lang sein.

9.1.3 Verlassen von EDIT

Um die Arbeit mit EDIT zu beenden, setzen Sie den Befehl W (Wind up) ein. EDIT spult sich bis zum Dateiende der Quelldatei durch, kopiert diese in die Zieldatei und steigt aus. Geben Sie keine mit TO angegebene Datei an, benennt EDIT die interne Ausgabedatei mit dem Namen der FROM-Datei.

EDIT kann Befehle aus verschiedenen Quellen erhalten. Im einfachsten Fall nimmt EDIT die Befehle direkt von der Tastatur; dies wird als primäre Befehlsstufe bezeichnet. EDIT kann darüber hinaus Befehle von anderen Quellen, wie zum Beispiel Befehlsdateien oder auch WITH-Dateien, annehmen.

Mit dem Befehl C können Sie Befehlsdateien während der Arbeit von EDIT aufrufen, innerhalb der Befehlsdateien können Sie dann weitere aufrufen, so daß jede verschachtelte Befehlsdatei zu einer eigenständigen Befehlsstufe wird. EDIT stoppt die Ausführung der Befehle in der Befehlsdatei, wenn das Dateiende erreicht wird oder der Befehl Q gefunden wurde. Findet EDIT ein Q in einer Befehlsdatei oder wird das Dateiende erreicht, bricht EDIT die Ausführung der Befehle der Datei ab und kehrt zur vorhergehenden Befehlsstufe zurück. Findet EDIT einen Q-Befehl in einer verschachtelten Befehlsdatei, kehrt EDIT zur Ausführung der Befehle in der Befehlsdatei, die jeweils eine Stufe höher liegt, zurück. Beenden Sie das Editieren auf der primären Befehlsstufe durch Eingabe eines Q-Befehls oder findet EDIT in einer WITH-Datei ein Q, dann beendet EDIT die Arbeit in gleicher Weise, als ob Sie den Befehl W verwendet hätten.

Der Befehl STOP beendet EDIT sofort, also ohne weiteres Abarbeiten der Datei. Dabei ist besonders zu beachten, daß EDIT keine der noch im Speicher stehenden Zeilen in die Zieldatei schreibt. Deklarieren Sie nur das FROM-Argument, überschreibt EDIT die Quelldatei nicht mit der unvollständig editierten Datei. Der Befehl STOP sollte nur dann verwendet werden, wenn das veränderte File nicht gebraucht wird.

EDIT legt eine interne Sicherheitskopie der bearbeiteten Datei in der Datei :T/ED-BACKUP ab, wenn Sie die Arbeit mit den Befehlen W oder Q beenden. Diese Backup-Datei verbleibt im Speicher, bis EDIT ein zweites Mal aufgerufen und wieder beendet wird. Dann überschreibt EDIT die Datei mit dem neuen Backup. Verwenden Sie den Befehl STOP, wird diese Datei nicht erstellt.

9.1.4 Ein kombiniertes Beispiel: alle Befehle auf einmal

Mit den bisher beschriebenen Befehlen können alle wichtigen Arbeiten innerhalb eines Files durchgeführt werden. Dieser Abschnitt nun stellt ein Beispiel vor, das einige dieser Befehle verwendet. Der Text, der im Beispiel den Editier-Befehlen in Kursivschrift folgt, ist Kommentar. Diese Kommentare dürfen nicht mit eingegeben werden; EDIT läßt in Befehlszeilen keine Kommentare zu.

Tippen Sie bitte folgenden Quelltext ab. Verwenden Sie den ED, die Zeilennummern bitte nicht mit eingeben:

```
1  Früher, wo ich sehr unerfahren
2  und bescheidener als heute
3  Hatten meine höchste achtung
4
5  Später tref ich auf den Weide
6  außer mir noch andere Kälber
7  und nun schätze ich, sozusagen
8  erst dich selber.
```

Führen Sie nun folgende Änderungen durch (die Reihenfolge der Befehle ist wichtig!):

```
M E/wo/da/ ;E /sehr //
```

1. Zeile, »wo« durch »da« ersetzen; »sehr« löschen

```
N; A /bescheidener /war /;A L//,/
```

nächste Zeile, »war« einsetzen; nach dem ersten Leerzeichen von rechts ein Komma an fügen

```
N; E /H/h/; E L /a/A/
```

das erste »H« durch »h« ersetzen, das letzte »a« durch »A« ersetzen.

```
F P//; I
andere Leute
Z
```

vor der Leerzeile einsetzen.

```
N; E /tref/traf/; E /den/der/
```

Nächste Zeile; Teile austauschen

```
M6; 2(A L//,/; N)
```

Zu Zeile 6; das letzte Leerzeichen durch ein »,« ersetzen; eine Zeile weiter; dort ebenfalls ersetzen

```
F B/erst/; E/d/m/
```

Zeile suchen, die mit »erst« beginnt; »d« durch »m« ersetzen

```
W
```

geändertes File speichern, EDIT beenden

Man erhält den folgenden Text (mit neuen Zeilennummern):

```
1  Früher, da ich unerfahren
2  und bescheidener war als heute,
3  hatten meine höchste Achtung andere Leute.
4
5  Später traf ich auf der Weide
6  außer mir noch andere Kälber,
7  und nun schätze ich, sozusagen,
8  erst mich selber.
```

Experimentieren Sie ruhig mit diesem Quelltext. Dabei werden Sie feststellen, daß es verschiedene Wege zu einer Lösung gibt. (Die Verse stammen übrigens von Wilhelm Busch.) So kann zum Beispiel die letzte Zeile auch so eingegeben werden:

```
E/d/m/
```

Das Resultat bleibt gleich, da diese Zeile durch das »N« in der vorhergehenden schon zur aktuellen wurde.

9.2 Vollständige Beschreibung des Zeileneditors

Im ersten Abschnitt dieses Kapitels haben Sie sich mit den grundlegenden Funktionen von EDIT vertraut gemacht. Damit können Sie schon alle normalerweise vorkommenden Arbeiten mit EDIT durchführen. Dieses Kapitel enthält eine vollständige Beschreibung aller Möglichkeiten von EDIT. Diesen Abschnitt werden Sie nur brauchen, wenn Probleme mit EDIT auftauchen oder Sie sehr komplizierte Änderungen an einer Datei vornehmen wollen.

Im einzelnen wird beschrieben:

- Befehlssyntax
- Befehlsfiles, Quellfile und Ausgabefile
- Arbeiten mit EDIT
- Funktionelle Gruppen von EDIT-Befehlen
- Zeilenfenster
- String-Bearbeitung in der aktuellen Zeile
- Andere Befehle für die aktuelle Zeile
- Teile des Quellcodes lesen
- Schleifen

- Umfassende Eingriffe
- Anzeigen des Programmstatus
- Beenden eines EDIT-Ablaufes
- Verändern der aktuellen Zeile
- Abbrechen der interaktiven EDITierung

9.2.1 Befehlssyntax

EDIT-Befehle bestehen aus einem Befehlsnamen, ohne oder mit einem oder mehreren Argumenten. Ein oder mehrere Leerzeichen zwischen Befehlsnamen und dem ersten Argument, zwischen Argumenten ohne Zeichenketten und zwischen Befehlen sind erlaubt. Ein Leerzeichen zwischen diesen Argumenten ist nur notwendig, um aufeinanderfolgende zu trennen, die sonst als eine Einheit behandelt werden; zum Beispiel zwei Zahlen. EDIT erkennt, daß ein Befehl beendet ist:

- wenn RETURN gedrückt wird
- wenn EDIT das Ende der Argumente eines Befehls erreicht
- wenn EDIT auf ein Semikolon (;) stößt
- wenn EDIT auf das Zeichen »)« stößt

Klammern setzen Sie, um Befehlsgruppen zu bilden.

Befehle, die in einer Eingabezeile auftreten, trennen Sie durch ein Semikolon (;). Dies ist nur notwendig, wenn ein Befehl eine variable Anzahl an Argumenten besitzt. EDIT nimmt immer die ausführlichste Form eines Befehls an. Sie können Befehle in Groß- und Kleinschrift eingeben.

9.2.1.1 Befehlsnamen

Ein Befehlsname ist entweder eine Reihe von Buchstaben oder ein einzelnes, spezielles Zeichen, wie zum Beispiel #. Ein alphabetischer Befehlsname endet mit einem Zeichen, das kein Buchstabe ist. Nur die ersten vier Buchstaben des Namens werden berücksichtigt. Ein oder mehrere Leerzeichen können zwischen dem Befehlsnamen und seinen Argumenten auftreten. EDIT verlangt mindestens ein Leerzeichen, wenn ein Argument, das mit einem Buchstaben beginnt, einem alphanumerischen Namen folgt.

9.2.1.2 Argumente

Die folgenden Abschnitte beschreiben die sechs verschiedenen Arten von Argumenten, die Sie zusammen mit EDIT-Befehlen verwenden können:

- Zeichenketten
- qualifizierte Zeichenketten
- Suchausdrücke
- Zahlen
- Einstellwerte
- Befehlsgruppen

9.2.1.3 Zeichenketten

Eine Zeichenkette ist eine Folge von bis zu 80 Zeichen, die von Abgrenzern eingeschlossen sind. Leere (Null-)Zeichenketten sind erlaubt (eine Null-Zeichenkette ist eine Zeichenkette ohne Zeichen, das heißt, die Abgrenzer schließen nichts ein, wie zum Beispiel //). Zeichen, die Sie zum Begrenzen einer Zeichenkette verwenden, dürfen innerhalb dieser Zeichenkette selbst nicht auftreten. Der beendende Abgrenzer kann weggelassen werden, wenn die Befehlszeile unmittelbar darauf endet.

Folgende Zeichen können als Abgrenzer verwendet werden:

```
/  ü  .  +  -  ,  ?  :  *
```

Dies sind also die üblichen Interpunktionszeichen, außer dem Semikolon (;), und die vier arithmetischen Operatoren. Hier einige Beispiele für Zeichenketten:

```
/A/
*Isartal-Sperre*
??
+Zeichenkette ohne abschließenden Abgrenzer
```

9.2.1.4 Mehrfach-Zeichenketten

Befehle mit zwei Zeichenketten als Argumente müssen den gleichen Abgrenzer für beide Zeichenketten verwenden. Der Abgrenzer zwischen den beiden Argumenten wird nur einmal eingegeben. Hierzu ein Beispiel mit dem Befehl A:

```
A /König/Der Rote/
```

In diesem Befehl deklariert die zweite Zeichenkette den einzufügenden Text. Lassen Sie diese Zeichenkette wegfallen, setzt EDIT eine Null-Zeichenkette ein. Tun Sie dies bei den Befehlen A und B, geschieht nichts, da EDIT angewiesen wird, nichts nach oder vor der ersten Zeichenkette einzufügen. Lassen Sie dagegen die zweite Zeichenkette nach einem E-Befehl fortfallen, löscht EDIT die erste Zeichenkette.

9.2.1.5 Qualifizierte Zeichenketten

Befehle, die nach Bezügen suchen, entweder in der aktuellen Zeile oder in der gesamten Quelldatei, bezeichnen den Bezug durch qualifizierte Zeichenketten. Eine qualifizierte Zeichenkette ist eine Zeichenkette, der kein, ein oder mehrere Qualifikatoren vorausgehen. Die Qualifikatoren sind einzelne Zeichen, die in jeder Reihenfolge verwendet werden können. Zum Beispiel:

```
BU /Abc/
```

Leerzeichen brauchen Sie zwischen den Qualifikatoren nicht einzusetzen. Eine Liste von Qualifikatoren beenden Sie durch ein beliebiges Abgrenzer-Zeichen. Die verfügbaren Qualifikatoren sind:

```
B(Anfang)
E(Ende)
L(Links oder Letztes)
P(Vorher)
U(Großschrift)
```

9.2.1.6 Suchausdrücke

Befehle, die nach einer einzelnen Zeile in der Quelldatei suchen, erwarten einen Suchbegriff als Argument. Ein Suchbegriff ist eine qualifizierte Zeichenkette. Der folgende Befehl sucht zum Beispiel eine Zeile, die mit der Zeichenkette »Weltall« beginnt:

```
F B/Weltall/
```

9.2.1.7 Zahlen

Eine Zahl ist eine Folge dezimaler Ziffern. Zeilennummern sind eine spezielle Zahlenform und müssen stets größer als null sein. Tritt eine Zeilennummer auf, können statt dessen auch die Zeichen ».« und »*« erscheinen. Ein Punkt steht für die aktuelle Zeile, ein Stern für die letzte Zeile der Quelldatei. Der folgende Befehl weist EDIT zum Beispiel an, zum Ende der Quelldatei zu springen:

```
M*
```

9.2.1.8 Einstellwerte

Befehle, die EDIT-Einstellungen verändern, erwarten ein einzelnes Zeichen als Argument. Dieses Zeichen muß ein Plus (+) oder ein Minus (–) sein.

```
V-
```

Das Minus (–) in diesem Befehl weist EDIT an, die Verifizierung einzustellen. Wird dann »V+« eingegeben, schaltet EDIT die Verifizierung wieder an. In diesem Fall können »+« und »D« als »an« und »aus« verstanden werden.

9.2.1.9 Befehlsgruppen

Um eine Anzahl eigenständiger EDIT-Befehle zu einer Befehlsgruppe zusammenzufassen, können Sie die Befehle in Klammern setzen. Die folgende Zeile sucht zum Beispiel die Zeile, in der der Begriff »Adler« erscheint, und ersetzt ihn durch »Großer Vogel«.

```
(F/Adler/; E/Adler/Großer Vogel/)
```

Befehlsgruppen können nicht länger als eine Eingabezeile sein. Geben Sie mehr Befehle ein, nimmt EDIT nur die bis zum Zeilenende an. EDIT findet dann die geschlossene Klammer am Ende der Zeile nicht und gibt die Fehlermeldung aus:

```
Unmatched parenthesis
```

Beachten Sie bitte, daß Sie Klammern nur verwenden müssen, wenn eine Befehlsgruppe mehr als einmal ausgeführt werden soll.

9.2.1.10 Wiederholung der Befehle

Vielen Befehlen kann eine Zahl vorangestellt werden, die EDIT anweist, diesen Befehl zu wiederholen. Die Zahl darf kein Vorzeichen haben und nur aus dezimalen Ziffern bestehen. Zum Beispiel:

```
26N
```

Geben Sie die Ziffer Null ein, führt EDIT den Befehl unendlich oft aus, das heißt, natürlich nur, bis das Dateiende erreicht ist. Der folgende Befehl ersetzt zum Beispiel bis zum Dateiende jedes »Dum« durch »Dee«:

```
0(E /Dum/Dee/;N)
```

Auch Befehlsgruppen können in gleicher Weise wie einzelne Befehle wiederholt werden:

```
0(F/hadlich/; E/hadlich/handlich/)
```

9.2.2 Programmablauf

Dieser Abschnitt beschreibt die Abläufe bei der Arbeit mit EDIT. Er gibt Auskunft darüber, woher Eingaben kommen und wohin Ausgaben gehen, was auf dem Bildschirm erscheint und was nach dem Ablauf von EDIT in der bearbeiteten Datei steht.

9.2.2.1 Das Eingabe-Zeichen (Prompt)

Arbeiten Sie interaktiv mit EDIT, erscheint ein Prompt, sobald EDIT bereit ist, neue Befehle auszuführen. Interaktiv wird gearbeitet, wenn Befehle von der Tastatur kommen und Ausgaben auf den Bildschirm geschrieben werden. Hat der letzte Befehl eine Ausgabe auf dem Bildschirm verursacht, gibt EDIT keinen Prompt aus.

Ist die Verifizierung aktiviert, aktualisiert EDIT die aktuelle Zeile,

- wenn die aktuelle Zeile nicht bereits verifiziert wurde
- wenn seit der letzten Verifizierung in der Zeile irgendwelche Änderungen gemacht wurden
- wenn die Lage des aktiven Fensters verändert wurde.

Wird die aktuelle Zeile von EDIT nicht verifiziert, zeigt der Zeileneditor einen Doppelpunkt (:), um darauf hinzuweisen, daß er zur Ausführung neuer Befehle bereit ist. Dieser Doppelpunkt ist das Eingabe-Zeichen von EDIT. EDIT gibt jedoch keine Prompts aus, wenn Sie Zeilen einfügen.

9.2.2.2 Die aktuelle Zeile

Während EDIT Zeilen aus einer Quelldatei liest oder sie in eine Zieldatei schreibt, wird die Zeile, die gerade bearbeitet wird, zur aktuellen Zeile. Jeder eingegebene Befehl arbeitet mit der aktuellen Zeile. EDIT fügt neue Zeilen vor der aktuellen Zeile ein. Beginnt man mit dem Editieren mit EDIT, ist zunächst die erste Zeile der Quelldatei die aktuelle Zeile.

9.2.2.3 Zeilennummern

EDIT erkennt jede Zeile in der Quelldatei anhand einer nur einmal vergebenen Zeilennummer. Diese ist kein Teil der in der Datei gespeicherten Informationen. EDIT vergibt diese Nummern durch Abzählen der Zeilen während des Einlesens. Neu eingefügte Zeilen erhalten keine Zeilennummern.

EDIT unterscheidet zwischen Originalzeilen aus dem Quelltext und Nicht-Originalzeilen. Originalzeilen sind Zeilen aus dem Quelltext, die nicht verändert oder getrennt wurden. Nicht-Originalzeilen wurden neu eingefügt oder durch Trennen erzeugt. Befehle, die Zeilennummern als Argument haben, können nur auf die Quellzeilen verweisen. EDIT springt vorwärts und bis zu einer bestimmten Grenze rückwärts. Die Richtung hängt davon ab, ob die eingegebene Zeilennummer größer oder kleiner als die aktuelle ist. EDIT überspringt oder löscht (auf Wunsch) neu eingegebene Zeilen auf der Suche nach angegebenen Quellzeilen. Geben Sie an Stelle einer Zeilennummer einen Punkt ein, beziehen sich alle Befehle auf die aktuelle Zeile, ob Quellzeile oder nicht. Ein Beispiel dazu finden Sie in Abschnitt 9.1.2.6 »Löschen ganzer Zeilen«.

Mit dem Befehl »=« werden die Zeilen umnumeriert. Dadurch erhalten neu eingegebene Zeilen Zeilennummern, werden also Quellzeilen gleichgestellt. Der folgende Befehl ändert zum Beispiel die aktuelle Zeilennummer auf 15, alle folgenden bis zum Dateiende werden angepaßt.

```
=15
```

Wird nach dem Befehl »=« keine Nummer angegeben, so zeigt EDIT folgende Meldung an:

```
Number expected after =
```

9.2.2.4 Qualifizierte Zeichenketten

Um den Bereich anzugeben, in dem EDIT suchen soll, verwenden Sie qualifizierte Zeichenketten. EDIT akzeptiert die Null-Zeichenkette, es vergleicht nur mit der angegebenen Suchposition. Die Suchposition ist, falls nicht anders angegeben, der Anfang einer Zeile. Geben Sie keine zusätzlichen Qualifikatoren an, findet EDIT die Zeichenkette an einer beliebigen Stelle in einer Zeile. Qualifikatoren deklarieren zusätzliche Bedingungen zur Eingrenzung der Suche. EDIT erlaubt die fünf Qualifikatoren B, E, L, P und U:

B sucht nur am Anfang der Zeile. Dieser Qualifikator darf nicht zusammen mit E, L oder P auftreten.

E sucht am Ende der Zeile. Dieser Qualifikator darf nicht mit B, L oder P auftreten. Erscheint E mit einer Null-Zeichenkette, entspricht das dem Zeilenende. Das heißt, das Leerzeichen am Zeilenende wird gesucht.

L durchsucht eine Zeile entgegen der normalen Richtung von rechts nach links. Tritt eine Zeichenkette in einer Zeile mehrmals auf, stellt dieser Qualifikator sicher, daß die letzte Zeichenkette statt der ersten gefunden wird. L darf nicht zusammen mit B, E oder P verwendet werden. Geben Sie L mit einer Null-Zeichenkette an, wird das Leerzeichen am Ende der Zeile gefunden.

P sucht nach der Zeile, die nur den als Argument angegebenen String enthält. P darf nicht mit B, E oder L auftreten. P mit einer Null-Zeichenkette als Argument sucht eine Leerzeile.

U setzt Groß- und Kleinschrift gleich; das heißt, die Zeichenkette kann aus jeder Kombination von Groß- und Kleinbuchstaben bestehen. Wird U zum Beispiel mit folgender Zeichenkette verwendet:

```
/HUMPAtäterä/
```

werden auch die beiden folgenden Zeile gefunden:

```
humpaTÄTERÄ
HUMpatätERÄ
```

9.2.2.5 Abspeichern der editierten Zeilen

EDIT schreibt bearbeitete Zeilen nicht unmittelbar in die Zieldatei, sondern speichert sie in einem als Puffer definierten Bereich im Hauptspeicher ab. Hat EDIT den verfügbaren Speicher verbraucht, schreibt es die ersten Zeilen aus diesem Puffer der Reihe nach in die Zieldatei. Bis zu dem Zeitpunkt, an dem EDIT eine bestimmte Zeile in die Zieldatei schreibt, kann man auf sie zurückgreifen und sie wieder zur aktuellen Zeile machen.

Teile der Ausgabe können Sie übrigens auch in andere Zieldateien als in die hinter TO angegebene schreiben.

9.2.2.6 Handhabung des Dateiendes

Stößt EDIT auf das Ende der Quelldatei, wird eine fiktive Datei-Endzeile erzeugt. Diese Datei-Endzeile trägt die nächsthöhere Zeilennummer. Diese Zeile erscheint mit der Zeilennummer »*« auf dem Bildschirm.

Befehle zum Ändern dieser aktuellen Zeile oder Sprünge zu einer weiteren Zeile erzeugen eine Fehlermeldung. Im Gegensatz dazu gibt EDIT keine Fehlermeldung aus, wenn das Dateiende während der Ausführung einer unendlich wiederholten Gruppe erreicht wird.

9.2.3 Funktionelle Einteilung der EDIT-Befehle

Dieser Abschnitt enthält, nach ihrer Funktion geordnet, alle EDIT-Befehle. Eine Zusammenfassung und eine alphabetische Liste folgen am Ende des Kapitels.

Die folgenden Beschreibungen verwenden Schrägstriche (/) als Abgrenzer; Abgrenzer sind Zeichen zum Begrenzen von Zeichenketten. Befehlsnamen werden in Großschrift dargestellt. Die Argumente werden in Kleinschrift angegeben, wie in der folgenden Tabelle beschrieben:

Bezeichnung	Beschreibung
a,b	Zeilennummer (oder ».« oder »*«)
cg	Befehlsgruppen
m,n	Nummern
q	Qualifikatoren-Liste (evtl. leer)
se	Suchausdruck
s,t	Zeichenketten
sw	Einstellwert (+ oder -)
/	Zeichenketten-Abgrenzer

Tabelle 9.1: Variable zur Beschreibung von Befehlen

Befehlsbeschreibungen, die die hier aufgelisteten Bezeichnungen verwenden, zeigen die Syntax eines Befehls. Sie sind keine Beispiele für das, was Sie eingeben müssen.

9.2.3.1 Auswahl einer aktuellen Zeile

Diese Befehle haben keine weitere Funktion als die Bestimmung einer neuen aktuellen Zeile. EDIT schreibt diese Zeile in den Pufferspeicher, sobald sie bearbeitet wurde. Weitere Informationen zu diesem Thema finden Sie in Abschnitt 9.1 EDIT – Ein zeilenorientierter Texteditor.

M nimmt eine Zeilennummer, einen Punkt oder einen Stern als Argument an. Wird in diesem Fall die oben beschriebene Notation verwendet, lautet die korrekte Syntax also:

```
Ma
```

Ma springt in der Quelldatei zur Zeile »a« vor oder zurück. Es können nur Quellzeilen angesprochen werden.

```
M+
```

läßt die letzte von der Datei gelesene Zeile zur aktuellen werden. M+ liest alle Zeilen, die momentan im Speicher gehalten werden und macht die letzte zur aktuellen Zeile.

```
M-
```

macht die letzte Zeile im Pufferspeicher zur aktuellen Zeile. EDIT geht also soweit wie möglich zurück.

```
N
```

springt zur nächsten Zeile der Quelldatei vor. Ist die aktuelle Zeile die letzte Zeile der Quelldatei, erscheint keine Fehlermeldung. Die Datei-Endzeile wird angezeigt. Geben Sie dann wiederum den Befehl N ein, gibt EDIT eine Fehlermeldung aus.

```
P
```

springt in die vorhergehende Zeile. Sie können mehrere Zeilen zurückspringen, indem Sie P wiederholt oder vor einer Zahl angeben. Die Zahl muß der Anzahl an Zeilennummern entsprechen, die zurückgesprungen werden soll.

```
F
```

findet die gesuchte Zeile durch den Suchausdruck »se«. Die Suche beginnt in der aktuellen Zeile und geht vorwärts durch die gesamte Quelldatei. Ein F-Befehl ohne Argument verwendet den bei der letzten Suchoperation angegebenen Suchbegriff. Die Syntax für den Befehl F lautet:

```
F se
```

Die Syntax für BF (finde rückwärts) lautet:

```
BF se
```

BF verhält sich wie F, sucht jedoch, ausgehend von der aktuellen Zeile, zum Anfang der Quelldatei, bis er eine Zeile findet, die den gesuchten Ausdruck enthält.

9.2.3.2 Einfügen und Löschen einer Zeile

Befehle können neben ihrer Hauptfunktion eine neue aktuelle Zeile bestimmen. Solche Befehle, gefolgt von einzufügendem Text, müssen die letzten Befehle einer Befehlszeile sein. Der einzufügende Text muß sich in den Zeilen unter dem Befehl befinden und wird mit »Z« auf einer eigenen Zeile beendet. Der Befehl Z wird auch zum Ändern des Zeichens für Dateiende verwendet. EDIT nimmt das Zeichen in Großschrift und Kleinschrift an. Der folgende Befehl fügt zum Beispiel den neuen Text vor »a« ein.

Ia(einzufügender Text, soviele Zeilen wie nötig)Z

Zur Erinnerung: »a« kann eine Zeilennummer, ein Punkt (für die aktuelle Zeile) oder ein Stern (für die letzte Zeile der Quelldatei) sein. Lassen Sie »a« fort, fügt EDIT den neuen Text vor der aktuellen Zeile ein; a wird dann die neue aktuelle Zeile.

```
I/s/
```

fügt den Inhalt der Datei »s« (s steht für Zeichenkette) vor der aktuellen Zeile ein.

```
Ra b
(zu ersetzender Text)
Z
```

Der Befehl R entspricht D;I. Die zweite Zeilennummer muß größer oder gleich der ersten sein. Lassen Sie die zweite Zeilennummer weg, ersetzt EDIT nur eine Zeile. Das heißt, b=a. Werden beide Nummern weggelassen, wird die aktuelle Zeile ersetzt. Die Zeile nach b wird zur neuen aktuellen Zeile.

Der Befehl D (Delete) löscht alle Zeilen von »a« bis einschließlich »b«. Wird die zweite Zeilennummer weggelassen, wird nur die angegebene Zeile gelöscht; das heißt, b=a. Werden beide Nummern weggelassen, wird die aktuelle Zeile gelöscht. Die Zeile, die auf »b« folgt, wird zur neuen aktuellen Zeile. Die Syntax für D lautet:

```
Da b
```

Der Befehl DF weist EDIT an, alle Zeilen ab der aktuellen Zeile zu löschen, bis die Zeile mit dem gesuchten Ausdruck gefunden ist. Diese Zeile wird dann zur neuen aktuellen Zeile. Ein DF-Befehl ohne Argument sucht den zuletzt eingegebenen Suchbegriff.

Die Syntax des Befehls DF (Delete and Find) lautet:

```
DF se
```

9.2.4 Zeilenfenster

EDIT bearbeitet normalerweise eine komplette aktuelle Zeile. Darüber hinaus können aber auch Zeilenteile definiert werden, die dann als Zeilenfenster bezeichnet werden. Dieser Abschnitt beschreibt, wie die Befehle zum Definieren eines solchen Fensters eingesetzt werden.

9.2.4.1 Das operationale Fenster

EDIT untersucht gewöhnlich alle Zeichen einer Zeile, wenn es nach einem String sucht. Es kann jedoch ein Zeilenfenster definiert werden, so daß die Suche nach einem Zeichen am Anfang des Fensters und nicht am Anfang einer Zeile beginnt. In allen Beschreibungen von EDIT-Befehlen bedeutet der Anfang der Zeile stets auch den Anfang des operationalen Fensters, sofern ein solches Fenster definiert ist.

Wenn EDIT eine aktuelle Zeile neu aufbaut, weist es mit einem Zeiger, dem Größer-Zeichen »>«, auf den Beginn des operationalen Fensters der aktuellen Zeile hin. Zum Beispiel:

```
26.
Dies ist Zeile 26.
   >
```

Das operationale Fenster enthält die Zeichen rechts vom Zeiger: »Dies ist Zeile 26.«. EDIT läßt den Indikator weg, falls es sich am Anfang der Zeile befindet.

Die folgenden Befehle zum Bewegen des Fensters stehen zur Verfügung:

```
>
```

bewegt den Zeiger um ein Zeichen nach rechts.

```
<
```

bewegt den Zeiger um ein Zeichen nach links.

```
PR
```

Der Pointer Reset setzt den Zeiger an den Anfang der Zeile.

```
PA q/s/
```

Der Befehl Point After setzt den Zeiger so, daß das erste Zeichen im Fenster das erste Zeichen nach der Zeichenkette »s« ist.

```
PA L//
```

bringt beispielsweise den Zeiger zum Zeilenende.

```
PB
```

Der Befehl PB (Point Before) ähnelt dem Befehl PA, nur daß die Zeichenkette selbst im Fenster steht. Die Syntax für PB lautet:

```
PB q/s/
```

9.2.4.2 Verändern einzelner Zeichen in der aktuellen Zeile

Die folgenden Befehle bewegen den Zeiger ein Zeichen nach rechts, nachdem der erste Buchstabe in einen Groß- oder Kleinbuchstaben verwandelt wurde. Ist das erste Zeichen kein Buchstabe oder steht er bereits in der gewünschten Schriftart, entsprechen diese Befehle »>«.

$ ändert zu Kleinschrift.
% ändert zu Großschrift.

Der Befehl _ (Unterstreichungsstrich) macht das erste Zeichen im Fenster zu einem Leerzeichen und bewegt den Zeiger eine Stelle nach rechts.

löscht das erste Zeichen im Fenster. Der im Fenster verbleibende Rest rutscht ein Zeichen nach links, während der Zeiger stehenbleibt und auf das nächste Zeichen der Zeile verweist. Dieser Befehl entspricht genau dem folgenden (wobei »s« das erste Zeichen des Fensters ist):

```
E/s//
```

Um diesen Befehl zu wiederholen, kann auch eine Zahl vorangestellt werden:

```
5#
```

löscht die nächsten fünf Zeichen im Fenster, entspricht also der Eingabe von »#####«. Geben Sie eine Zahl ein, die größer oder gleich der im Fenster vorhandenen Zeichen ist, löscht EDIT den Inhalt des gesamten Fensters.

Kombinationen aus >, %, $, _ und #-Befehlen können Sie zum zeichenweisen Editieren verwenden. Die Befehle erscheinen unter den Zeichen, die sie betreffen. Dazu folgendes Beispiel:

```
ein männLEIN stehttim walde,, ganz    stilllund STUMM
```

Dies Zeile wird nach Eingabe der folgende Befehle

```
%_%_$$$$____%_>>#_>>#____>$$$$$
```

zu

```
Ein Männlein steht im Walde, ganz still und stumm
```

Der Zeiger bleibt danach unter dem letzten »m« stehen.

9.2.5 Verändern von Zeichenketten in der aktuellen Zeile

Änderungen in der aktuellen Zeile werden mit den grundlegenden Befehlen zur String-Bearbeitung oder mit einer Variante, dem Zeiger-zu-Befehl, durchgeführt. Beide werden in den nächsten zwei Abschnitten beschrieben.

9.2.5.1 Grundlegende Operationen mit Zeichenketten

Zum Verändern von Teilen der aktuellen Zeile stehen drei sehr ähnliche Befehle zur Verfügung. Die Befehle A, B und E setzen ihr zweites Argument nach (A=After), vor (B=Before) oder anstelle (E=Exchange) des ersten Arguments. Das erste Argument kann zusätzlich durch einen Qualifikator bestimmt sein. Lautet die aktuelle Zeile:

```
Der Bäcker backt Kuchen
```

und Sie geben dann ein:

```
E U/Bäcker/Vater/  Austausch
B/Kuchen/gute  /   Einfügen der Zeichenkette vor
A L//;/            Einfügen der Zeichenkette nach
```

ändert sich die Zeile zu

```
Der Vater backt gute Kuchen;
```

9.2.5.2 Der Null-String

Die leere oder Null-Zeichenkette (//) kann nach jedem Zeichenketten-Befehl stehen. Geben Sie die Null-Zeichenkette als zweite Zeichenkette eines E-Befehls ein, löscht EDIT die erste Zeichenkette aus der Zeile. Findet EDIT die erste Zeichenkette, bewirkt die Null-Zeichenkette als zweites Argument bei den Befehlen A oder B nichts; anderenfalls wird eine Fehlermeldung ausgegeben. Bei allen Befehlen entspricht die Eingabe der Null-Zeichenkette als erstes Argument der ersten Suchposition. Die ursprüngliche Suchposition ist die aktuelle Zeigerposition (zu Beginn der Zeilenanfang), wenn nicht die Qualifikatoren E oder L eingegeben wurden. Zum Beispiel:

```
A//Bäcker/
```

setzt den Text »Bäcker« nach »Nichts«, das heißt an den Anfang der Zeile. Der folgende Befehl

```
A L//Bäcker
```

setzt zum Beispiel den String »Bäcker« an das Ende der Zeile, sozusagen nach dem letzten »Nichts«.

9.2.5.3 Die Zeiger-Variante

Die Befehle AP (insert After and Point), BP (insert Before and Point) und EP (Exchange and Point) haben zwei Zeichenketten als Argumente und arbeiten wie A, B und E. Dabei bleibt der Zeiger jedoch an der letzten Stelle der beiden Argumente stehen. So entspricht:

```
AP/s/t/
```

der Folge

```
A/s/t/; PA/st/
```

während

```
BP/s/t/
```

der Folge

```
B/s/t/; PA/ts/
```

entspricht.

Das folgende Beispiel

```
2EP U/diedel/Diddel/
```

ändert den Text

```
diedeldie und DIEDELdum
```

in

```
Diddeldie und Diddeldum
```

wobei der Zeiger vor »dum« stehen bleibt.

9.2.5.4 Löschen von Teilen der aktuellen Zeile

Die beiden Befehle DTA (Delete Till After) und DTB (Delete Till Before) löschen vom Zeilenanfang (oder vom Zeiger) bis zu einer angegebenen Zeichenkette. Um von einem angegebenen Punkt zum Zeilenende zu löschen, verwenden Sie die Befehle DFA (Delete From After) und DFB (Delete From Before). Lautet die aktuelle Zeile zum Beispiel:

```
All des Königs Pferde und all des Königs Mannen
```

und der Befehl

```
DTB L/Königs/
```

wird getippt, ändert sich der Text in

```
Königs Mannen
```

Der folgende Befehl hingegen

```
DTA/Pferde /
```

ändert den Text in

```
und all des Königs Mannen
```

9.2.6 Weitere Befehle für die aktuelle Zeile

In diesem Abschnitt wird erklärt, wie Befehle wiederholt werden, wie die aktuelle Zeile aufgetrennt und Zeilen miteinander verbunden werden.

Immer wenn EDIT einen Befehl zum Ändern einer Zeichenkette ausführt (zum Beispiel A, B oder E), wird diese Zeile zur aktuellen. Um den Befehl ein weiteres Mal auszuführen, geben Sie ein Apostroph (»'«) ein. Dieser Befehl hat kein Argument. Er verwendet die Argumente des letzten A-, B- oder E-Befehls.

Achtung! Verwenden Sie Befehle wie diese, können unerwartete Dinge geschehen:

```
E/Schloß/Ritter/; 4(«; E/Bauer/Königin/)
```

Die zweite und weitere Ausführungen des Befehls »'« verweisen auf eine andere als die erste Ausführung. Das obige Beispiel tauscht zweimal »Schloß« und »Ritter« und siebenmal »Bauer« und »Königin«, anstatt »Schloß« und »Ritter« einmal auszutauschen und danach viermal »Schloß« durch »Ritter« und »Bauer« durch »Königin« zu ersetzen.

9.2.6.1 Trennen und Verbinden von Zeilen

EDIT ist in erster Linie ein Zeileneditor. Die meisten EDIT-Befehle wirken nicht über die Zeilengrenzen hinaus. Dieser Abschnitt beschreibt nun Befehle, die eine Zeile aufteilen oder zwei oder mehr aufeinanderfolgende Zeilen miteinander verbinden.

Um eine Zeile vor einem angegebenen Bezugspunkt aufzuteilen, wird der Befehl SB verwendet. Die Syntax für SB lautet:

```
SB q/s/
```

SB erkennt Qualifikatoren, wie hier durch q dargestellt, und eine Zeichenkette /s/. SB teilt die aktuelle Zeile vor dem Bezugspunkt, den Sie durch Qualifikatoren und die Zeichenkette angeben. EDIT schreibt den ersten Teil der Zeile in den Puffer und macht den verbliebenen Rest zur neuen aktuellen Zeile, die jedoch keine Quellzeile ist. Um eine Zeile nach dem Bezugspunkt aufzuteilen, wird der Befehl SA verwendet. Die Syntax von SA ist:

```
SA q/s/
```

SA akzeptiert bei Bedarf Qualifikatoren q und eine Zeichenkette /s/. SA teilt die aktuelle Zeile nach dem Bezugspunkt, den Sie durch Qualifikatoren und Zeichenkette angeben.

Um zwei Zeilen zu verketten, verwenden Sie den Befehl CL. Dessen Syntax lautet:

```
CL/s/
```

CL bezieht bei Bedarf eine Zeichenkette, die oben als /s/ dargestellt wird, ein. CL (Concatenate Line) bildet eine neue aktuelle Zeile durch Verbinden der aktuellen Zeile mit der angegebenen Zeichenkette und der nächsten Zeile aus der Quelldatei, und zwar in dieser Reihenfolge. Ist die gesuchte Zeichenkette ein Null-String, braucht sie nicht angegeben werden. Das Aufteilen und Verbinden von Zeilen soll anhand eines Beispiels aufgezeigt werden:

```
Fischers Fritz aß frische Fische; Frische
Fische aß Fischers
Fritz.
```

Dieser Vers ist etwas verstümmelt. Die Zeilen müssen noch in Form gebracht werden. Ist die erste Zeile die aktuelle, ändern die folgenden Befehle

```
SA /; /; 2CL/ /
```

die erste Zeile zu

```
Fischers Fritz aß frische Fische;
```

und machen die zweite (die nun »Frische Fische aß Fischers Fritz« enthält) zur aktuellen Zeile.

9.2.7 Teile der Quelldatei lesen

Alle folgenden Befehle lesen Zeilen von der Quelldatei und schreiben die dabei abgearbeiteten Zeilen in den Pufferspeicher oder die normale Ausgabedatei. Da diese Befehle in der Regel interaktiv angewandt werden, das heißt, der Bildschirm wird aktualisiert, werden sie Type-Befehle genannt. Sie können jedoch auch zur Ausgabe in eine Datei verwandt werden. Nachdem EDIT einen dieser Befehle ausgeführt hat, wird die zuletzt getippte Zeile, das heißt die zuletzt auf dem Bildschirm angezeigte, zur neuen aktuellen Zeile.

Die Syntax des Befehls T (Type) lautet:

```
Tn
```

Tn tippt n Zeilen. Lassen Sie n fort, wird bis zum Ende der Quelldatei ausgegeben. Die Ausgabe können Sie jederzeit mit [Ctrl]+[C] unterbrechen.

Hinweis: Steht irgendwo im Handbuch ein Plus-Zeichen zwischen zwei Tasten, bedeutet das, daß sie zur gleichen Zeit gedrückt werden müssen. [Ctrl]+[C] heißt deshalb, daß [C] eingegeben wird, während [Ctrl] gedrückt ist.

Die erste Zeile, die EDIT nach »T« ausgibt, ist die aktuelle. Der folgende Befehl gibt, zum Beispiel beginnend mit der nächsten Zeile, die »Sein oder nicht sein« enthält, sechs Zeilen auf den Bildschirm aus:

```
F /Sein oder nicht sein/; T6
```

Beachten Sie bitte, daß Sie das »S« vom ersten »Sein« in Großschrift eingeben müssen, wenn die richtige Zeile gefunden werden soll.

Der Befehl TP gibt die Zeilen im Pufferspeicher aus. TP (Type Previous) entspricht dem in EDIT ausgeführten Befehl M, gefolgt von der Ausgabe der Zeilen bis zur aktuellen. Die Syntax lautet:

```
TP
```

Der Befehl TN gibt alle Zeilen aus, die sich noch im Hauptspeicher des EDIT befinden. TN (Type Next) gibt alle Zeilen auf dem Bildschirm aus, die auch mit dem Befehl P erreicht werden können. Mehr über den Befehl P finden Sie in Abschnitt 9.1.1 dieses Handbuchs. Der Vorteil des TN-Befehls liegt darin, daß alle ausgegebenen Zeilen mit den Befehlen P und BF angesprungen werden können.

Die Syntax des Befehls TL (Type with Line numbers) lautet:

```
TL n
```

TLn tippt n Zeilen wie T, im Unterschied dazu werden Zeilennummern hinzugefügt. Da eingefügte und aufgeteilte Zeilen keine Zeilennummern haben, gibt EDIT bei ihnen »++++« aus. Zum Beispiel:

```
20   O Bruder Montague, komm heraus
++++ wir leben hier in Saus und Braus
```

Die Quellzeile beginnt mit »O Bruder« und hat eine Zeilennummer. Die nicht zur Quelldatei gehörende eingefügte Zeile beginnt mit »++++«. Zur Erinnerung: Mit dem Befehl »=« können Sie Zeilen numerieren, auch wenn sie nicht zur Quelldatei gehören.

9.2.8 Verwaltung von Befehls-, Eingabe- und Ausgabedateien

EDIT verwendet vier Dateiarten:

- Befehlsdatei
- Eingabedatei
- Ausgabedatei
- Verifizierungsdateien

Ist EDIT gestartet, kann die Verifizierungsdatei mit keinem Befehl verändert werden. Mehr über die Verifizierungsdatei finden Sie in Abschnitt 9.1.1 dieses Handbuchs. Der folgende Abschnitt beschreibt Befehle, die Befehls-, Eingabe- und Ausgabedateien verändern, die Sie beim Start von EDIT erstellt oder geöffnet haben.

9.2.8.1 Befehlsdateien

EDIT liest Befehle vom Terminal oder aus einer Datei, die Sie mit der Option WITH angegeben haben. Um Befehle aus einer anderen Datei zu lesen, verwenden Sie den Befehl C. Laut Muster lautet die Syntax für den Befehl C:

```
C /s/
```

wobei die Zeichenkette »s« für einen Dateinamen steht. Da Amiga-DOS den Schrägstrich (/) zum Trennen von Dateinamen benutzt, sollten Sie Punkte oder andere Symbole als Begrenzer verwenden. Ein Symbol, das sich in der Zeichenkette befindet, sollte nicht als Abgrenzer benutzt werden. Hat EDIT alle Befehle einer Datei ausgeführt (oder ist Q gefunden worden), schließt EDIT die Datei und greift wieder auf den Befehl zurück, der dem Befehl C folgt. Die folgende Befehlszeile zum Beispiel liest und führt die Befehle in der Datei »:T/XYZ« aus:

```
C .:t/xyz.
```

9.2.8.2 Eingabedateien

Um den gesamten Inhalt einer Datei an einer bestimmten Stelle in die Quelldatei einzufügen, verwenden Sie die Befehle I und R. Diese Befehle wurden bereits in Abschnitt 9.1.2.7 dieses Kapitels besprochen.

In Abschnitt 9.1.1 wird auch beschrieben, wie EDIT aufgerufen wird. Dort wird die Quelldatei als FROM-Datei angegeben. Mit FROM als Schlüsselwort können Sie jedoch auch eine Datei angeben, aus der weitere Zeilen wie Quelltext eingelesen werden sollen. FROM hat folgende Syntax:

```
FROM /s/
```

wobei die Zeichenkette »s« ein Dateiname ist. FROM ohne Argument liest die ursprüngliche Quelldatei noch einmal ein. Wird FROM ausführt, bleibt die aktuelle Zeile gleich, die nächste Zeile kommt aus der neuen Quelldatei.

EDIT schließt eine Datei nicht, wenn die Datei nicht mehr aktuell ist, es können weitere Zeilen aus dieser Quelldatei gelesen werden, indem sie wieder angewählt wird. Um eine Ausgabedatei, die in EDIT geöffnet wurde, zu schließen, um sie daraufhin zur Eingabe zu öffnen, müssen Sie CF (Close File) verwenden. CF hat folgende Syntax:

```
CF /s/
```

Dabei stellt »s« wieder einen Dateinamen dar. Wird die Arbeit mit EDIT beendet, werden alle Dateien geschlossen.

Öffnen Sie eine Datei, so beginnt EDIT am Anfang dieser Datei zu lesen oder zu schreiben. Wird eine Datei mit CF geschlossen und wieder geöffnet, so beginnt EDIT wieder mit der ersten Zeile der Datei und nicht mit der Zeile, an der EDIT sich befand, als die Datei geschlossen wurde.

Ein Beispiel zum Binden von Zeilen aus zwei Dateien mit dem Befehl FROM:

Befehl	Wirkung
M10	arbeitet die Zeilen 1 bis 9 aus der Quelldatei ab.
FROM .XYZ.	wählt neue Eingabe an, Zeile 10 bleibt aktuell.
M6	arbeitet Zeile 10 aus FROM ab und holt die Zeilen 1 - 5 aus XY
Z.	
FROM	Datei FROM wird wieder angewählt.
M14	arbeitet Zeile 6 aus XYZ und Zeile 11 bis 13 aus FROM ab.
FROM .XYZ.	Datei FROM wird wieder angewählt.
M*	arbeitet Zeile 14 aus FROM und den Rest aus XYZ ab.
FROM	Datei FROM wird wieder angewählt.
CF .XYZ.	XYZ wird geschlossen.
M*	der Rest von FROM (Zeile 15 bis Dateiende) wird abgearbeitet.

9.2.8.3 Ausgabedateien

EDIT schickt seine Ausgabe in eine TO-Datei. Die Ausgabe erfolgt jedoch nicht unmittelbar. EDIT hält eine bestimmte Anzahl von Zeilen in einem Pufferbereich im Hauptspeicher. Diese Zeilen sind vorhergehende aktuelle Zeilen oder Zeilen, die EDIT bereits abgearbeitet hat, bevor es die gegenwärtige aktuelle Zeile erreichte. Die Anzahl der Zeilen, die EDIT im Speicher behalten kann, hängt von der Puffergröße ab, die Sie beim Aufruf von EDIT zugewiesen haben. Weil EDIT diese Zeilen zurückhält, hat es die Fähigkeit, in der Quelldatei rückwärts zu gehen.

Um die Ausgabe in eine andere als die beim Start angegebene Datei zu schreiben, wird ebenfalls der Befehl TO verwendet. TO hat die Syntax:

```
TO .s.
```

Dabei ist »s« der Dateiname.

Führt EDIT einen TO-Befehl aus, schreibt es den Pufferspeicher in die jeweilige Datei, wenn die Ausgabedatei geöffnet ist. EDIT schließt eine Ausgabedatei nicht, wenn sie nicht weiter aktuell ist. Wird die Datei wieder angewählt, können weitere Zeilen hineingeschrieben werden. Das folgende Beispiel zeigt, wie Sie die Ausgabe zwischen der Hauptrichtung TO und einer anderen Richtung XYZ aufteilen können.

Befehl	Wirkung
M11	schreibt die Zeilen 1 bis 10 in Richtung TO.
TO.XYZ.	.aktiviert die Ausgabedatei XYZ.
M21	schreibt die Zeilen 11 bis 21 in Richtung XYZ.
TO	
M31	schreibt die Zeilen 21 bis 30 in Richtung TO.
TO.XYZ.	
M41	schreibt die Zeilen 31 bis 40 in Richtung XYZ.
TO	

Wollen Sie eine Datei erneut benutzen, so müssen Sie sie vorher schließen. Der folgende Befehl schließt die Datei mit dem als Argument angegebenen Dateinamen:

```
CF .Dateiname.
```

Diese Ein-/Ausgabe-Befehle werden angewandt, wenn ein Teil der Quelldatei von oben nach unten bewegt werden soll. Die folgende Befehlsliste zeigt dies am Beispiel:

Befehl	Wirkung
TO .:T/1.	Ausgabe an eine Hilfs-Datei :T/1.
1000N	schreibt 1000 Zeilen in die Hilfs-Datei.
TO	kehrt zur eigentlichen Ausgabedatei zurück.
CF .:T/1.	schließt Hilfs-Datei :T/1.
I2000.:T/1.	öffnet :T/1 als Eingabedatei und liest die Zeilen wieder ein.

Verwenden Sie CF bei Dateien, die nicht mehr bearbeitet werden sollen, wird der von EDIT benötigte Speicherplatz verringert.

9.2.9 Schleifen

Bei vielen Befehlen können Sie eine Wiederholung bewirken, indem Sie ihnen eine Zahl (ohne Vorzeichen) voranstellen. Zum Beispiel:

```
24N
```

Befehlsgruppen können in ähnlicher Form mehrmals ausgeführt werden. Zum Beispiel:

```
12(F/hadlich/; E/hadlich/handlich/)
```

Geben Sie als Zahl null (0) ein, wird der Befehl oder die Befehlsgruppe so oft wiederholt, bis EDIT das Ende der Quelldatei erreicht.

9.2.10 Umfassende Operationen

Umfassende Operationen sind Operationen, die automatisch durchgeführt werden, während EDIT die Quelldatei nach unten abarbeitet. Umfassende Operationen werden mit speziellen

Befehlen begonnen und beendet. Diese Befehle werden im folgenden Abschnitt beschrieben. Achtung! Beim Rückwärtsgehen durch die Quelldatei sollten Sie darauf achten, daß alle umfassenden Befehle vorher gelöscht werden. Aktive umfassende Befehle können eine Menge erledigter Arbeit zunichte machen!

9.2.10.1 Festlegen von umfassenden Änderungen

Drei Befehle, GA, GB und GE, stehen für Zeichenketten-Änderungen zur Verfügung. Ihre Syntax lautet wie folgt:

```
GA q/s/t/
GB q/s/t/
GC q/s/t/
```

Diese Befehle führen einen A-, B- oder E-Befehl wie gewohnt an jeder Stelle durch, an der die Zeichenkette »s« in einer neuen aktuellen Zeile erscheint. Sie werden jeweils in der Zeile ausgeführt, die gerade aktuell ist.

G-Befehle bearbeiten ihren ersetzenden Text nicht nochmals. Der folgende Befehl ist deshalb keine Endlos-Schleife, hätte aber keine sichtbaren Auswirkungen in irgendeiner Zeile.

```
GE/rote Rüben/rote Rüben/
```

Das einzige Ergebnis der Änderung wäre die Verifizierung der Zeilen mit dem Inhalt »rote Rüben«.

EDIT führt die umfassenden Änderungen in jeder neuen aktuellen Zeile in der Reihenfolge aus, in der Sie sie angegeben haben.

9.2.10.2 Aufheben von umfassenden Änderungen

Der Befehl REWIND bricht alle umfassenden Operationen ab. Mit dem Befehl CG (Cancel Global) können einzelne umfassende Befehle abgebrochen werden.

Wurde eine umfassende Operation mit den Befehlen GA, GB oder GE gestartet, ist der Operation eine Kenn-Nummer zugewiesen, die in die Verifizierungsdatei ausgegeben wird; zum Beispiel G1. Das Argument von CG ist die Nummer der umfassenden Operation, die abgebrochen werden soll. Wird CG ohne Argument ausgeführt, löscht EDIT alle umfassenden Operationen.

9.2.10.3 Aufschub umfassender Änderungen

Einzelne umfassende Änderungen können mit dem Befehl DG (Disable Global) vorübergehend aufgehoben und mit EG (Enable Global) wieder eingesetzt werden. Auch diese Befehle benötigen die Kenn-Nummer der jeweiligen Operation als Argument. Lassen Sie das Argument fort, schaltet EDIT alle umfassenden Operationen an oder aus.

9.2.11 Anzeigen des Programmstatus

Zwei Befehle, die mit SH (SHow) beginnen, geben Informationen über den Status von EDIT an die Verifizierungsdatei aus. Der Befehl SHD (SHow Data) hat folgende Form:

```
SHD
```

Er zeigt gespeicherte oder voreingestellte Informationen, wie zum Beispiel den letzten Suchbegriff. Der Befehl SHG (SHow Globals) hat die Form:

```
SHG
```

Er zeigt die aktuellen umfassenden Befehle, zusammen mit ihrer Kenn-Nummer. Ebenso wird die Anzahl der durchgeführten Änderungen zu jeder umfassenden Operation angegeben.

9.2.12 Beenden eines EDIT-Ablaufs

Der Befehl W (Windup) wird zum Vorspulen der restlichen Quelldatei verwendet. Wird das Dateiende erreicht und sind alle Dateien geschlossen und der Puffer-Speicher geleert, wird EDIT dadurch beendet.

Rufen Sie EDIT nur mit dem FROM-Dateinamen auf, so benennt EDIT die zeitweilige Ausgabedatei, die während der Arbeit erzeugt wurde, mit dem gleichen Namen wie dem des Originals. Das heißt, die Ausgabedatei bekommt den Namen der FROM-Datei, während der ursprüngliche Inhalt der FROM-Datei in der Datei »:T/EDIT-BACKUP« abgelegt wird. Diese Sicherheitskopie (Backup) der Datei ist natürlich nur bis zu einem erneuten Ablauf von EDIT verfügbar. W ist unzulässig, wenn die Ausgabe momentan nicht auf TO gerichtet ist.

Der Befehl STOP beendet EDIT sofort. Es werden keine weiteren Ein- oder Ausgaben durchgeführt. Die Quelldatei wird nicht überschrieben. Die Eingabe von STOP bietet die Sicherheit, daß in der Eingabedatei keine Änderungen durchgeführt werden. Q unterbricht die Ausführung der aktuellen Befehlsdatei. EDIT liest Befehle normalerweise von der Tastatur ein, jedoch können auch Befehlsdateien mit dem Schlüsselwort WITH oder dem Befehl C angegeben werden. Wurde die Ausführung der aktuellen Befehlsdatei abgebrochen, kehrt EDIT zur vorhergehenden Eingabe-Ebene zurück. Ein Q in der ersten Ebene bewirkt das gleiche wie W.

9.2.13 Verifizieren der aktuellen Zeile

Eine Zeile wird automatisch verifiziert (neu ausgegeben), wenn:

- ein neuer Befehl eingegeben wurde, der Veränderungen an der aktuellen Zeile bewirkte,
- eine Zeile noch nicht ausgegeben wurde, seit sie aktuell ist, und zu einer veränderten, aber nicht ausgegebenen Zeile zurückgesprungen wird,
- eine Fehlermeldung ausgegeben wird.

In den ersten drei Fällen wird nur verifiziert, wenn die Option V auf EIN gestellt ist. Der folgende Befehl ändert die Einstellung von V.

```
V sw
```

V steht auf EIN, wenn beim Programmstart nicht anders angegeben. Interaktiv wird gearbeitet, wenn Befehle und Verifizierungen mit dem Terminal verbunden sind.

Um die Verifizierung der aktuellen Zeile zu erreichen, verwenden Sie den Befehl ?.

```
?
```

Die Verifizierung wird automatisch ausgeführt, wenn die Option V eingeschaltet ist und der Inhalt der Zeile geändert wurde. Die Verifizierung besteht aus der Zeilennummer oder »++++«, wenn die Zeile nicht zur Quelldatei gehört, und dem Text auf der Zeile.

Eine andere Form der Verifizierung der aktuellen Zeile bietet der Befehl »!«. Er wird eingesetzt, um zusätzlich eine Statuszeile auszugeben. Die Syntax dieses Befehls lautet:

```
!
```

Der Befehl ! verifiziert die aktuelle Zeile mit Zeichenindikatoren. EDIT zeigt zwei Zeilen. Die erste ist die aktuelle Zeile. In der zweiten Zeile zeigt EDIT ein Minus-Zeichen unter all jenen Zeichen, die Großbuchstaben sind und ein Kennzeichen für Steuerzeichen. Alle anderen Stellen enthalten Leerzeichen. Zum Beispiel zeigt der Befehl ! unter einer Zeile, die einen Tabulatorsprung enthält, an:

```
4.
Diese Zeile enthält einen            Tabulatorsprung.
-    -                  T............-
```

9.2.14 Verschiedenartige Befehle

Dieser Abschnitt beschreibt all jene Befehle, die sich nicht in die zuvor beschriebenen Kategorien einfügen. Er beschreibt, wie das Beendigungszeichen verändert wird, wie nachfolgende Leerzeichen ausgeschaltet, Zeilen umnumeriert und die Quelldatei zurückgespult wird.

Um das Zeichen zum Abschluß einer Texteinfügung zu ändern, wird der Befehl Z verwendet. Er hat die folgende Form:

```
Z/s/
```

wobei /s/ für eine Zeichenkette steht. Diese kann bis zu 16 Zeichen lang sein. Groß- und Kleinbuchstaben sind erlaubt. Mit dem Qualifikator PU kann nach dem Zeichen gesucht werden. Das ursprüngliche Zeichen ist »Z«.

Um nachfolgende Leerzeichen (am Ende der Zeile) ein- oder auszuschalten, verwenden Sie den Befehl TR (TRailing spaces). Der Befehl TR hat die folgende Form:

```
TR sw
```

wobei sw für EIN (+) oder AUS (–) steht. EDIT unterdrückt gewöhnlich alle am Ende einer Zeile stehenden Leerzeichen. TR+ erlaubt sie in Eingabe- und Ausgabezeilen. Um Zeilen umzunumerieren, wird der Befehl »=« verwendet. Er hat folgende Syntax:

```
=n
```

wobei n für eine Nummer steht. »=n« ändert die Nummer der aktuellen Zeile zu n. Alle nachfolgenden, auch die nicht zur Quelldatei gehörenden Zeilen werden umnumeriert. In der gleichen Weise werden nach »=« alle vorhergehenden Zeilen in der Ausgabereihe als nicht zur Quelldatei gehörig markiert. Wird EDIT beendet, erhalten alle Zeilen, ob Quellzeile oder nicht, und auch solche Zeilen, die zuvor schon umnumeriert wurden, eine neue Nummer.

Um die Quelldatei zurückzuschreiben, verwenden Sie den Befehl REWIND. Seine Syntax lautet:

```
REWIND
```

Dieser Befehl spult die Eingabedatei so zurück, daß Zeile 1 wieder zur aktuellen wird. Zuerst untersucht EDIT die restliche Quelldatei, um umfassende Operationen auszuführen. Dann werden alle Zeilen in die Zieldatei geschrieben, diese geschlossen und als neue Quelldatei wieder geöffnet. Die ursprüngliche Quelldatei wird geschlossen und eine temporäre Datei als Zieldatei verwendet. Alle umfassenden Operationen werden beendet. EDIT erkennt diesen Befehl bereits, wenn »REWI« eingetippt wird.

9.2.15 Abbrechen der interaktiven Editierung

Um Befehle abzubrechen, die nach einem String suchen, verwenden Sie [Ctrl]+[C]. Haben Sie den Suchbegriff falsch eingegeben, bricht [Ctrl]+[C] die Suche ab. Entsprechend gibt der Befehl T bis zum Ende der Quelldatei aus, wird aber mit [Ctrl]+[C] abgebrochen.

Nach Eingaben von [Ctrl]+[C] gibt EDIT folgende Meldung aus:

```
*** BREAK
```

und kehrt in den Eingabe-Modus zurück. Die aktuelle Zeile ist die, die sich im Speicher befand, als Sie [Ctrl]+[C] betätigt hatten.

9.3 Kurzübersicht

In dieser Liste werden als Abkürzungen verwendet:

Abkürz.	Beschreibung
qs	Qualifizierte Zeichenkette
t	Zeichenkette
n	Zeilennummer (oder ».« für aktuelle Zeile bzw. »*« für letzte Zeile)
sw	+ oder - (für »ein« und »aus«)

Befehle in Zeilenfenstern

Befehl	Funktion
<	Zeiger nach links setzen.
>	Zeiger nach rechts setzen.
#	Zeichen löschen.
$	Zeichen zu Kleinbuchstaben ändern.
%	Zeichen zu Großbuchstaben ändern.
_	Zeichen löschen und Leerzeichen einfügen.
PA qs	Bewegt Zeiger nach qs.
PB qs	Bewegt Zeiger vor qs.
PR	Bewegt Zeiger zum Zeilenanfang.

Befehle zum Zeilensprung

Befehl	Funktion
M n	Geht zu Zeile n.
M +	Geht zur ersten Zeile im Speicher.
M -	Geht zur letzten Zeile im Speicher.
N	Geht zur nächsten Zeile.
P	Geht zur vorhergehenden Zeile.
REWIND	Liest vorheriges Zielfile als Quellfile ein.

Befehle zum Suchen von Strings

Befehl	Funktion
F qs	Findet String qs.
BF qs	Wie F, sucht zum Anfang des Files.
DF qs	Wie F, löscht alle Zeilen bis zur gesuchten.

Befehle zum Ausgeben von Text

Befehl	Funktion
?	Zeigt aktuelle Zeile.
!	Zeigt Zeile und Statuszeile.
T	Gibt alle Zeilen bis zum Fileende aus.
T n	Gibt n Zeilen aus.
TL n	Gibt n Zeilen mit Zeilennummer aus.
TN	Gibt Zeilen aus, bis alle Zeilen im Puffer geändert wurden.
TP	M- und dann alle verfügbaren Zeilen ausgeben.
V sw	Schaltet Verifizierung ein oder aus.

Operationen in der aktuellen Zeile

Befehl	Funktion
A qs t	Setzt String t hinter qs ein.
AP qs t	Wie A, bewegt aber den Zeiger.
B qs t	Setzt String t vor qs ein.
BP qs t	Wie B, bewegt aber den Zeiger.
CL t	Verbindet aktuelle Zeile, String t und nächste Zeile zu einer Zeile.
D	Löscht die aktuelle Zeile.
DFA qs	Löscht Zeile ab dem Ende von qs.
DFB qs	Löscht Zeile ab dem Anfang von qs.
DTA qs	Löscht vom Anfang der Zeile bis hinter qs.
DTB	Löscht vom Anfang der Zeile bis vor qs.
E qs t	Ersetzt qs durch t.
EP qs t	Wie E, bewegt aber den Zeiger.
I	Setzt Text von der Tastatur vor der aktuellen Zeile ein.
I t	Übernimmt Text aus File t.
R	Ersetzt Text mit Text vom Terminal.
R t	Ersetzt Text mit Text aus File t.
SA qs	Trennt Zeile hinter qs.
SB qs	Trennt Zeile vor qs.

Umfassende Befehle

Befehl	Funktion
GA qs t	Wirkt wie A, jedoch im gesamten Quellfile.
GB qs t	Wirkt wie B, jedoch im gesamten Quellfile.
GE qs t	Wirkt wie E, jedoch im gesamten Quellfile.
CG n	Löscht umfassenden Befehl Nummer n (oder alle, wenn n nicht angegeben wird).
DG n	Setzt umfassenden Befehl Nummer n außer Kraft (oder alle, wenn n nicht eingegeben wird).
EG n	Aktiviert umfassenden Befehl Nummer n wieder (oder alle, wenn n nicht eingegeben wird).
SHG	Zeigt Informationen über alle umfassenden Befehle.

Ein- und Ausgabe-Befehle

Befehl	Funktion
FROM	Übernimmt Text aus Quelldatei.
FROM t	Übernimmt Text aus File t.
TO	Schreibt Text in Zieldatei.
TO t	Schreibt Text an File t.
CF t	Schließt File t.

Sonstige Befehle

Befehl	Funktion
«	Wiederholt vorangehenden A-, B- oder E-Befehl.
= n	Numeriert Datei neu durch, beginnend in der aktuellen Zeile mit n.
C t	Übernimmt Befehle aus File t.
H n	Setzt Stop-Zeiger in Zeile n (wenn n=*, wird Zeiger gelöscht).
Q	Beendet Befehlsebene, wie W, wenn in Ebene 1.
SHD	Zeigt Informationen über den Programmstatus.
STOP	Beendet EDIT unverzüglich.
TR sw	Erlaubt oder verbietet Leerzeichen am Zeilenende.
W	Beendet EDIT und schreibt Puffer in die Zieldatei.
Z t	Definiert t als Zeichen zum Abschluß eingefügten Textes.

10 MEMACS

MEMACS ist eine an den Amiga angepaßte Version des bildschirmorientierten Texteditors MicroEmacs. Dieser Editor ist auf fast jedem Computer vorhanden, und setzt einen gewissen Standard, auf den auch Commodore mit ihrem Amiga nicht verzichten wollte. Wem der MicroEmacs von einem anderen Computersystem her bekannt ist, kann auch den MEMACS auf dem Amiga ohne Probleme bedienen.

Mit dem MicroEmacs können gleichzeitig mehrere verschiedene Textdateien bearbeitet werden. Jede dieser Dateien muß vollständig in den Hauptspeicher des Amigas passen, da MEMACS die Datei in den Hauptspeicher lädt und alle Änderungen an der Kopie im Hauptspeicher vornimmt. Die Länge der Zeilen ist auf 80 Zeichen beschränkt, längere Zeilen werden nicht gekürzt, alle Zeichen jenseits des achtzigsten Zeichens werden nur nicht mehr angezeigt. Ein Dollarzeichen am Zeilenende zeigt diese Überlänge an.

MEMACS kann durch einen Doppelklick von der Workbench aus gestartet werden oder von einem SHELL-Fenster:

```
MEMACS [<Dateiname>] [GOTO <n>] [OPT W]
```

Dateiname gibt den Namen der Textdatei an, die bearbeitet werden soll. Mit dem Schlüsselwort GOTO läßt sich die Zeilennummer <n> angeben, an die der Cursor nach dem Laden gesetzt werden soll. OPT W öffnet das MEMACS-Fenster auf der Workbench.
MEMACS benutzt ein Workbenchfenster oder einen eigenen Screen, um seine Sichtfenster auf die verschiedenen Puffer dem Benutzer darzubieten. Wenn MEMACS mehrere Editorfenster darstellt, teilt er sein Workbenchfenster in mehrere vertikale Bereiche, in denen unabhängig voneinander die einzelnen Puffer bearbeitet werden können. Jeder dieser Sichtbereiche kann beliebig einem Puffer mit »Select-buffer« zugeordnet werden. Da MicroEmacs seine Ursprünge vor über zehn Jahren auf Rechnersystemen ohne Fensterverwaltung genommen hat, und für die Arbeit mit Textterminals optimiert wurde, nutzen seine Routinen die, erst mit der Einführung des Amigas und ähnlicher MC68000-Systeme sich verbreitenden, grafischen Oberflächen mit ihren Fenstern nicht aus. Diese Unabhängigkeit von rechnerspezifischen Eigenheiten hat aber auch zur weiten, systemübergreifenden Verbreitung dieses Editors beigetragen.

10.1 Einführung in das Editieren mit MEMACS

Editierkommandos werden durch Tastenkombinationen von der [Ctrl]- und der [Esc]-Taste gegeben, oder können mit der Maus in der Menüleiste ausgewählt werden. Bei jedem Befehl in der Menüleiste steht auch die Tastenkombination dabei, so daß mit steigender Übung auch die Tastaturkommandos verwendet werden können. Ein Dach (^) heißt, daß die [Ctrl]-Taste, [Esc], daß die [Esc]-Taste gedrückt werden muß.

Die Editierbefehle sind in der Menüleiste zu Gruppen gerordnet.

Project	Operationen zum Laden und Speichern von Textdateien.
Edit	Funktionen zum Bearbeiten eines vorhandenen Textes.
Window	Öffnen weitere Editorfenster für weitere Textdateien oder anderer Ausschnitte der Datei.
Move	Zum Bewegen im Text.
Line	Zum Kopieren einer Zeile, Öffnen oder Löschen einer neuen Zeile.
Word	Zum Bearbeiten von Worten.
Search	Suchfunktionen im Text.
Extras	Definition und Ausführen von Macros (programmierten Befehlssequenzen).

Eine neue Datei kann durch Starten von MEMACS mit dem Namen der neuen, noch nicht vorhandenen Datei angelegt werden. MEMACS öffnet seinen Screen oder sein Workbenchfenster und zeigt ein leeres Fenster, an dessen unterem Rand eine Statuszeile eingeblendet wird, die über den Namen der aktuell bearbeiteten Datei und die ausgeführte Aktion Auskunft gibt.

Nun kann ein Text, ein Programmtext oder ein Brief, eingegeben werden. Mit den Cursortasten kann man sich über den eingegebenen Text bewegen. Die Cursortasten in Verbindung mit [Shift] bewegen den Cursor an den jeweiligen Rand des Fensters. Ebenfalls kann der Cursor mit der Maus auf eine bestimmte Stelle im Text gesetzt werden, indem man sie mit der Maus anklickt. Mit [Del] kann das Zeichen unter dem Cursor gelöscht werden, mit [Backspace] das Zeichen links vom Cursor. Wenn der Text fertig ist, kann er mit [Ctrl]+[X] [Ctrl]+[S] oder »Save« aus dem Project-Menü gespeichert werden. Danach erscheint in der Statuszeile die Meldung »Wrote <n> lines«, um Auskunft darüber zu geben, das der Text von n Zeilen Länge in das aktuelle Verzeichnis gespeichert worden ist.

Mit dem Kommando [Esc]+[<] springt der Cursor an den Anfang der ersten Textzeile des Textes, [Esc]+[>] springt an das Ende des Textes. Mit [Esc]+[F] springt er ein Wort nach vorn. Mit [Ctrl]+[N] springt er in die nächste Zeile in die gleiche Spalte und mit [Ctrl]+[A] an den Anfang der nächsten Zeile.

Nach dem Drücken von [Ctrl]+[K] wird die aktuelle Zeile gelöscht und alle weiteren rücken eine Position auf, so daß die nächste Zeile zur aktuellen Zeile wird. [Ctrl]+[Y] setzt die eben gelöschten Zeilen an der Cursorposition wieder in den Text ein.

Um MEMACS zu verlassen, muß [Ctrl]+[X] [Ctrl]+[C] eingegeben werden. Es erscheint die Abfrage »Quit [y/n]«. Mit »y« kann der Editor verlassen werden.

Alle diese Befehle sind auch über die Pull-down-Menüs erreichbar. Wenn Befehle eine Eingabe benötigen, springt der Cursor in die unterste Bildschirmzeile und erwartet eine Eingabe.

10.2 Das Project-Menü

Rename [Ctrl]+[X]+[F]**:**
Der Name der Datei im Puffer soll geändert werden. Der neue Name der Datei kann optional zusammen mit dem Zugriffspfad auf das Verzeichnis angegeben werden, in das die Datei geschrieben werden soll.

Read-file [Ctrl]+[X] [Ctrl]+[R]**:**
Laden einer Datei. Soll die Datei aus einem anderen als dem aktuellen Verzeichnis geladen werden, muß der gesamte Zugriffspfad mit dem Namen angegeben werden.

Visit-file [Ctrl]+[X] [Ctrl]+[V]**:**
Eine weitere Datei soll zusätzlich in einen eigenen Puffer geladen werden. Die schon bearbeitete Datei wird davon nicht beeinflußt. Steht die angegebene Datei schon in einem Puffer von MEMACS, schaltet der Editor direkt zu diesem Puffer um, ohne die Datei ein zweites Mal zu laden.

Insert-file [Ctrl]+[X] [Ctrl]+[I]**:**
Der Inhalt der angegebenen Datei wird an der Position, an der der Cursor steht, in den vorhandenen Text eingefügt. Auch bei diesem Befehl muß der vollständige Pfad zur Datei eingegeben werden.

Save-file [Ctrl]+[X] [Ctrl]+[S]**:**
Der Inhalt des aktuellen Puffers wird unter dem angegebenen Dateinamen auf Diskette oder Festplatte abgespeichert. Ist kein Dateiname angegeben, in der Statuszeile wird dann kein Dateiname angezeigt, bricht MEMACS den Versuch zu speichern mit der Fehlermeldung »No File Name« ab. War der Schreibversuch der Datei erfolgreich, so meldet MEMACS, wieviele Textzeilen in die Datei geschrieben wurden.

Save-as-file [Ctrl]+[X] [Ctrl]+[W]**:**
Der aktuelle Textpuffer soll einen neuen Namen (und Zugriffspfad) erhalten, und unter diesem wie mit »Save-file« abgespeichert werden.

Save-mod [Ctrl]+[X] [Ctrl]+[M]**:**
Alle Puffer, deren Inhalt verändert wurde, sollen gespeichert werden. Dieser Befehl birgt die Gefahr, daß auch Dateien geschrieben werden, die probehalber verändert wurden, aber nicht wieder auf die Platte zurückgeschrieben werden sollten.

Save-exit [Ctrl]+[X] [Ctrl]+[F]**:**
Alle veränderten Puffer werden abgespeichert und der Editor verlassen. Dieser Befehl sollte mit der gleichen Vorsicht verwendet werden, wie »Save-mod«.

New-Cli [Ctrl]+[-]:
MEMACS kann ein SHELL-Fenster starten, von dem aus beliebig viele DOS-Befehle abgesetzt werden können. Der Editor ist, solange das SHELL-Fenster existiert, blockiert. Wenn das SHELL-Fenster mit ENDCLI geschlossen wird, kann das Arbeiten mit MEMACS normal an der Stelle fortgesetzt werden, an der unterbrochen wurde.

Cli-command [Ctrl]+[X]:
Ein DOS-Befehl kann ausgeführt werden, MEMACS zeigt die Ausgabe des Befehls in einem Editorfenster und Puffer. Der Puffer kann normal weiter verwendet oder einfach geschlossen werden.

Quit [Ctrl]+[C] **oder** [Ctrl]+[X] [Ctrl]+[C] **oder** [Esc] [Ctrl]+[C]:
Die Arbeit mit MEMACS soll beendet werden. Falls einer der Puffer verändert wurde, fragt MEMACS, ob sie wirklich den Editor verlassen wollen, ohne die Arbeit zu speichern:

Modified buffers exist, do you really want to exit? [y/n]

Wenn jetzt »y« [Return] für Ja eingegeben wird, werden die Änderungen nicht gespeichert, und sind unwiderruflich verloren, wird »n« [Return] für nein eingegeben, wird der Editor nicht verlassen, und es kann normal weitergearbeitet werden.

Wenn zum Beispiel ein älteres oder fremdes Programm als Grundlage für ein neues Projekt verwendet werden soll, ist es meist nicht beabsichtigt, die Vorlage verstümmelt abzuspeichern, so daß der Editor verlassen werden soll, ohne die Veränderungen an diesem Quelltext abzuspeichern.

Mit »List-buffers« aus dem Menü Edit kann man sich vor der Eingabe von »y« oder »n« eine Liste aller Puffer anzeigen lassen, die bearbeitet wurden. Die Namen der Puffer, die verändert wurden, sind mit einem Stern (*) gekennzeichnet. Aufgrund dieser Information kann entschieden werden, ob vergessen wurde, einen Puffer abzuspeichern.

10.3 Das Edit-Menü

Kill-region [Ctrl]+[W]:
Nachdem mit »Set mark« der Anfangspunkt eines Textabschnittes markiert wurde, kann der gesamte Ausschnitt mit »Kill-region« aus dem Text entfernt werden. Der ausgeschnittene Text wird in einem Zwischenspeicher abgelegt und kann entweder ignoriert werden, weil er gelöscht werden soll, oder an anderer Stelle wieder mit »Yank« in den Text eingesetzt werden.

In dem Zwischenspeicher werden aufeinanderfolgend alle Textabschnitte gespeichert, die aus einem der Puffer von MEMACS ausgeschnitten wurden. Wird das erste Mal »Yank« aufgerufen, wird der gesamte Textpuffer an die Cursorposition kopiert und der Textpuffer abgeschlossen. »Yank« kann nun beliebig oft nacheinander aufgerufen werden. Ruft man ein weiteres Mal »Kill-region« auf, so wird der bisherige Inhalt des Puffers verworfen und das Füllen des Puffers kann von neuem beginnen. Alle vorherigen Abschnitte gehen verloren.

Mit »Kill-region« können auch Abschnitte zwischen verschiedenen Dateien, die sich im Editor befinden, verschoben und beliebig dupliziert werden.

Yank [Ctrl]+[Y]**:**
kopiert den Inhalt des Zwischenspeichers von »Kill-region« ab der Cursorposition in den aktuellen Textpuffer.

Set-mark [Ctrl]+[@] **oder** [Esc]+[.]**:**
Die aktuelle Cursorposition wird als Bezugspunkt abgespeichert. Wird der Cursor danach weiterbewegt, kann man immer innerhalb desselben Textpuffers mit »Swap-dot&mark« aus dem Move-Menü von der momentanen Cursorposition zur gespeicherten zurückkehren. Dabei legt der Editor die verlassene Cursorposition auf den Speicher, so daß mit einem weiteren Aufruf von »Swap-dot&mark« wieder dorthin gesprungen werden kann.

Der Textabschnitt zwischen der gespeicherten und der momentanen Cursorposition dient aber auch zum Markieren eines Bereichs, der mit »Kill-region« ausgeschnitten werden soll.

Copy-region [Esc]+[w]**:**
Wie »Kill-region«, schneidet den Textabschnitt jedoch nicht aus, sondern kopiert ihn in den Puffer.

Upper-region [Ctrl]+[X] [Ctrl]+[U]**:**
Alle Buchstaben in dem mit »Set-mark« und der aktuellen Cursorposition markierten Abschnitt werden in Großbuchstaben umgewandelt.

Lower-region [Ctrl]+[X] [Ctrl]+[L]**:**
Alle Buchstaben in dem mit »Set-mark« und der aktuellen Cursorposition markierten Abschnitt werden in Kleinbuchstaben umgewandelt.

List-buffers [Ctrl]+[X] [Ctrl]+[B]**:**
Das Editorfenster des aktuellen Puffers wird in zwei Hälften geteilt, und in dem neuen Teilfenster eine Liste aller Puffer ausgegeben, die zur Zeit im Editor angelegt sind. Nur der Puffer, in dem das Listing angezeigt wird, erscheint nicht in der Liste. Dieser Puffer kann normal weiterverwendet oder einfach gelöscht werden.

Die Liste ist in vier Spalten unterteilt, in denen Informationen über die Datei angezeigt werden. In der ersten Spalte steht ein Stern (*), wenn eine Veränderung an der Datei vorgenommen wurde. In der zweiten Spalte wird die Größe der Datei in Zeilen angegeben. In der dritten Zeile erscheint der Name des Puffers, er entspricht immer dem (auf fünfzehn Buchstaben) gekürzten Namen der Datei, nur die zuerst eingeladene Datei erhält den Namen »main«. In der vierten Spalte wird der komplette Zugriffspfad auf die Datei angezeigt.

Die Statuszeile zeigt bei »List-buffers« »MEMACS-[List]«, und das Fenster, in dem die Namen der Puffer aufgelistet werden, erhält den Namen »[List]«. Dieser Name bleibt erhalten, auch wenn eine neue Datei in dieses Fenster eingeladen wird. Die Anzeige in diesem Fenster kann genauso wie in allen anderen Fenstern editiert und abgespeichert werden.

Man kann das Listingfenster auch weiterhin auf dem Bildschirm belassen, wenn man am Text in anderen Puffern weiterarbeitet. Ändert man den Namen eines der Puffer, wird die Anzeige im Listingfenster nicht mehr aktualisiert, das Listingfenster wurde zu einem normalen Editorfenster. Um aktuell Informationen über die Puffer zu erhalten, muß »List-buffers« erneut aufgerufen werden. Es verwendet dann dasselbe Fenster, falls dieses noch nicht geschlossen wurde.

Beispiel für ein Listing der Puffernamen:

```
C     Size       Buffer              File
-----------------------------------------------
      127012     ldos-befehle.asc     ldos-befehle.asc
      6040 1     einleitung.asc      leinleitung.asc
      59797      main                leinfuehrung.asc
```

Select-buffer [Ctrl]+[X]+[b]**:**
Holt einen anderen Puffer in das aktuelle Editorfenster zum Anschauen und Bearbeiten. In der Eingabezeile unter der Statuszeile fragt MEMACS dann nach dem Namen des Puffers, der bearbeitet werden soll.

Wird ein Name angegeben, der nicht einem schon vorhandenen Puffer zugeordnet ist, legt MEMACS einen neuen Puffer des angegebenen Namens an. Nur [Return] weist MEMACS an, »Select-buffer« zu ignorieren. Diesem neuen Puffer ist noch kein Dateiname zugeordnet. Deshalb muß dies zum Abspeichern entweder mit »Rename« geschehen, oder der Puffer muß mit der Funktion »Save-as-file« abgespeichert werden.

Insert-buffer [Esc]-[Ctrl]-[Y]**:**
Der Inhalt eines Puffers, dessen Namen von MEMACS abgefragt wird, soll oberhalb der aktuellen Zeile in den bearbeiteten Text eingesetzt werden. In der Eingabezeile am unteren Fensterrand erscheint dazu die Eingabeaufforderung:

```
Insert buffer:
```

Kill-buffer [Ctrl]+[X]+[k]**:**
Der Name des zu löschenden Puffers wird abgefragt, und danach der Puffer und der von ihm belegte Speicherplatz freigegeben. Bei Speicherknappheit ist es manchmal sinnvoll, einzelne Puffer zu löschen, um genügend Platz zu schaffen, damit eine größere Datei in den Speicher paßt, die sonst nicht editiert werden könnte.

Justify-buffer [Ctrl]+[X]+[J]**:**
Der Text im aktuellen Puffer wird im Blocksatz formatiert. Das heißt, am Zeilenanfang werden alle führenden Leerzeichen und Tabulatoren entfernt, und der Text wird bündig zu den aktuellen Rändern gesetzt.

Redisplay [Ctrl]+[L]**:**
Abhilfe, wenn im Editorfenster nur noch Müll dargestellt wird. Der Fensterinhalt wird neu aufgebaut. Diese Funktion ist ein Überbleibsel von Implementierungen des Editors auf Rechnersystemen, wo von einem Zentralrechner die Daten über eine teils lange und unsichere Leitung zu einem Terminal am Arbeitsplatz übertragen werden. Dabei auftretende

Übertragungsfehler konnten die Anzeige unlesbar machen. Diese Funktion sollte auf einem Amiga unnötig sein und dient zur Kompatibilität zu diesen anderen MicroEmacs-Versionen.

Quote-char Ctrl+Q oder Ctrl+X+q:
MicroEmacs interpretiert viele Kontrollzeichen als Menüaufrufe, beziehungsweise Funktionsaufrufe von Editorfunktionen. Diese Tastenkombinationen haben in Textdateien jedoch auch ihre spezielle Bedeutung. Sollen also diese Steuerzeichen in den Textpuffer eingegeben werden, muß zuerst Ctrl+Q und dann die Kontrollkombination eingegeben werden.

Für Ctrl+L also:

```
Ctrl+Q  Ctrl+L
```

Indent Ctrl+J:
Setzt ein Zeilenende und rückt die nächste Zeile genausoviele Leerzeichen wie die aktuelle Zeile ein. Der Cursor steht unter dem ersten Textzeichen der verlassenen Spalte.

Transpose Ctrl+T:
Tauscht die Position des Zeichens, auf dem der Cursor momentan steht, mit dem Zeichen links davon.

Cancel Ctrl+G:
Mit »Cancel« kann ein gestarteter Befehl während seiner Ausführung gestoppt werden.

10.4 Das Window-Menü

One-window Ctrl+X+1:
Das aktuelle Fenster wird in maximal möglicher Größe auf dem Schirm dargestellt. Alle anderen Fenster werden geschlossen. Der Inhalt der in diesen Fenstern angezeigten Puffer wird hierdurch nicht verändert.

Split-window Ctrl+X+2:
Das aktuelle Fenster wird in zwei Hälften unterteilt und stellt in beiden neuen Hälften den gleichen Ausschnitt des Puffers dar. Beide Fenster können unabhängig voneinander editiert werden. Alle Änderungen beziehen sich immer auf den gemeinsamen Puffer. So können zwei verschiedene, möglicherweise weit auseinanderliegende Teile eines Puffers gleichzeitig bearbeitet werden.

Next-window Ctrl+X+n:
Wechsel des aktuellen Fensters. Außer durch Anklicken mit der Maus kann auch mit »Next-window« das aktuelle Fenster gewechselt werden. Der Cursor springt zum nächsten unterhalb liegenden Fenster. Alle Fenster können reihum durchgeschaltet werden.

Prev-window Ctrl+X+p:
Das oberhalb liegende Editorfenster wird zum aktuellen Fenster.

Expand-window [Ctrl]+[X]+[z]:
Das aktuelle Fenster wird um eine Zeile vergrößert, das nachfolgende Fenster dafür um eine Zeile verkleinert. Kein Fenster läßt sich über die durch das Workbenchfenster oder den Screen vorgegebene Größe vergrößern.

Shrink-window [Ctrl]+[X]+[Z]:
Verkleinert das aktuelle Fenster um eine Zeile und vergrößert dafür das nachfolgende Fenster.

Next-w-page +[Esc][Ctrl]+[V]:
Blättert das nachfolgende Fenster um eine Seite weiter. Man braucht also beim Lesen eines zweiten Fensters zum Weiterblättern nicht das Editorfenster zu wechseln, da das aktuelle Fenster von »Next-w-page« nicht gewechselt wird. Ist nur ein Fenster vorhanden, wird der Text in diesem Fenster geblättert.

Prev-w-page +[Ctrl][X]+[v]:
Das vorhergehende Fenster wird um eine Seite zurückgeblättert.

10.5 Das Move-Menü

Top-of-buffer [Esc]+[<]:
Der Cursor wird an den Anfang des Puffers gesetzt.

End-of-buffer [Esc]+[]]:
Der Cursor wird auf das Ende des Puffers gesetzt.

Top-of-window [Esc]+[,]:
Der Cursor wird auf den Anfang der obersten Zeile im Editorfenster gesetzt.

End-of-window [Esc]+[.]:
Der Cursor wird auf das Ende der untersten Zeile im Editorfenster gesetzt.

Goto-line [Ctrl]+[X] [Ctrl]:
Nach Abfrage der gewünschten Zeilennummer springt der Cursor an die eingegebene Zeile im Puffer. Wird keine Zeilennummer angegeben, springt der Cursor in die erste Zeile des Puffers. Wird eine Zeilennummer eingegeben, die größer ist als die Nummer der vorhandenen Zeilen, springt der Cursor zur letzten vorhandenen Zeile.

Swap-dot&mark [Ctrl]+[X] [Ctrl]+[X]:
Wenn mit »Set-mark« eine Marke im Text gesetzt wurde, kann man zwischen der gesetzten Marke und der aktuellen Cursorposition hin- und herspringen. Wurde keine Marke gesetzt, so meldet MEMACS »No mark in this window«. Beim ersten Aufruf von »Swap-dot&mark« springt der Cursor zu der markierten Stelle, beim zweiten Aufruf kehrt der Cursor zu der verlassenen Position im Text zurück.

Next-page [Ctrl]+[V]:
Der Text wird um eine Seite, das heißt so weit nach vorn geblättert, daß die letzte Zeile, die vor dem Blättern im Editorfenster angezeigt wird, die erste Zeile der nächsten Seite wird. Der Cursor behält seine Position relativ zum Editorfenster bei und wird im Text auf eine weiter vorn liegende Position gesetzt.

Prev-page [Esc]+[v]:
Der Text wird eine Seite zurückgeblättert.

Next-word [Esc]+[f]:
Setzt den Cursor auf das nächste folgende Zeichen, das kein Buchstabe ist. Die Routine wertet allerdings auch alle nationalen Sonderzeichen (ä,ö,ü,ß, ...) nicht als Buchstaben.

Prev-word [Esc]+[b]:
Setzt den Cursor vor das erste Zeichen, das nicht als Buchstabe gewertet wird.

Scroll-up [Ctrl]+[Z]:
Der Text im Editorfenster wird um eine Zeile nach oben gerollt.

Scroll-down [Esc]+[z]:
Der Text wird im Editorfenster um eine Zeile nach unten gerollt.

10.6 Das Line-Menü

Open-line [Ctrl]+[O]:
Nach der aktuellen Zeile wird eine neue Zeile geöffnet. Die aktuelle Zeile wird an der Cursorposition geteilt. Alle auf den Cursor folgenden Zeichen werden in die neu geöffnete Zeile kopiert. Der Cursor bleibt an seiner Position in der alten Zeile stehen.

Kill-line [Ctrl]+[X] [Ctrl]+[D]:
Löscht die aktuelle Zeile aus dem Puffer und hängt den Inhalt an den Kill-puffer an. Siehe auch »Kill-region«.

Kill-to-eol [Ctrl]+[K]:
Der Text von der aktuellen Cursorposition bis zum Zeilenende wird gelöscht und in den Kill-puffer übertragen. Wenn sofort nach »Kill-to-eol« der Befehl »Yank« aufgerufen wird, wird nur die Zeile in ihrer alten Form wiederhergestellt.

Start-of-line [Ctrl]+[A]:
Der Cursor wird in die erste Spalte der aktuellen Zeile gesetzt.

End-of-line [Ctrl]+[E]:
Der Cursor wird auf das letzte Zeichen der aktuellen Zeile gesetzt. Ist die Zeile länger als achtzig Zeichen, wird dies optisch kenntlich gemacht, indem der Cursor auf das Dollarzeichen ($) am rechten Editorfensterrand gesetzt wird, das eine überlange Zeile kennzeichnet.

Next-line [Ctrl]+[N]**:**
Der Cursor wird um eine Zeile nach unten bewegt.

Previous-line [Ctrl]+[P]**:**
Der Cursor wird um eine Zeile nach oben bewegt. (Dieser Befehl ist wie der vorige ein Relikt aus einer Zeit, in der Computertastaturen noch keine Cursortasten besaßen.)

Line-to-top [Esc]+[!]**:**
Der Pufferinhalt wird nach oben gerollt, bis sich die aktuelle Zeile in der obersten Zeile des Editorfensters befindet.

Delete-blanks [Ctrl]+[X] [Ctrl]+[O]**:**
Von der aktuellen Zeile abwärts werden alle leeren Zeilen bis zur ersten Text enthaltenden Zeile gelöscht.

Show-line [Ctrl]+[X]+[=]**:**
Die aktuelle Cursorposition im Verhältnis zum Gesamttext wird in der Statuszeile ausgegeben. Nach dem Stichwort »Line« erscheint die aktuelle Zeile, nach »Column« die aktuelle Spalte, gefolgt von der Angabe in Prozent, ein wie großer Teil des Gesamttextes vor dem Cursor liegt.

10.7 Das Word-Menü

Delete-forw [Esc]+[d]**:**
Von der aktuellen Cursorposition bis zum nächsten Leerzeichen, Satzzeichen oder Tabulator werden alle Buchstaben gelöscht. Normalerweise kann man damit ein gesamtes Wort löschen, wenn der Cursor auf dem ersten Zeichen des Wortes steht, man ist jedoch auf diesen Fall nicht festgelegt. Der Cursor kann überall in der Zeile stehen und löscht dann bis zum nächsten Wortende. Steht man auf einem Leerzeichen, wird überhaupt nichts gelöscht.

Delet-back [Esc]+[h] **oder** [Esc]+[Del]**:**
Löscht von der Cursorposition an rückwärts bis zum ersten Leerzeichen.

Upper-word [Esc]+[u]**:**
Von der Cursorposition bis zum ersten folgenden Leerzeichen werden alle Buchstaben in Großbuchstaben umgewandelt.

Lower-word [Esc]+[l]**:**
Von der Cursorposition bis zum ersten folgenden Leerzeichen werden alle Buchstaben in Kleinbuchstaben umgewandelt.

Cap-word [Esc]+[c]**:**
Das Zeichen an der Cursorposition wird in einen Großbuchstaben umgewandelt, alle folgenden Zeichen bis zum Wortende werden in Kleinbuchstaben umgewandelt.

Switch-case [Esc]+[^]**:**
Von der aktuellen Cursorposition aus werden alle Klein- in Großbuchstaben geändert und umgekehrt.

10.8 Das Search-Menü

Search-forward [Ctrl]+[S] oder [Ctrl]+[X]+[s]:
MEMACS bietet die Möglichkeit, im Text nach Zeichenfolgen zu suchen. Dabei spielt die Groß- oder Kleinschreibung des Suchbegriffs keine Rolle, unabhängig davon werden alle textuellen Übereinstimmungen im Text gefunden. »Search forward« durchsucht beginnend bei der aktuellen Cursorposition den Text. In der Statuszeile wird der Suchtext abgefragt.

Search-backward [Ctrl]+[R] oder [Ctrl]+[X]+[r]:
Sucht von der aktuellen Cursorposition aus rückwärts.

Search-replace [Esc]+[r]:
Von der aktuellen Cursorposition aus vorwärts wird der Text nach dem angegebenen Suchbegriff durchsucht. Nachdem der Suchbegriff das erste Mal gefunden wurde, wird der Anwender nach einer Zeichenkette gefragt, die als Ersatz für die gefundene Zeichenkette in den Text eingefügt werden soll. Die Ersatzzeichenkette wird immer in der angegebenen Schreibweise eingefügt. MEMACS sucht bis zum Dateiende alle Zeichenfolgen, die dem Suchmuster entsprechen, und ersetzt sie durch die Ersatzzeichenfolge und meldet danach, wieviel Ersetzungen stattgefunden haben. (Replaced <n> occurrences).

Query-s-r [Esc]+[q]:
Sucht und ersetzt wie »Search-replace«, fragt aber bei jedem Auffinden des Suchmusters:

```
Change string [y/n/c/^G]?
```

Die Auswahl wird zwischen »y«, für ja, bitte ersetzen, »n«, für nein, nicht ersetzen, »c«, alle weiteren ersetzen und [Ctrl]+[G] für abbrechen gegeben.

Fence-match [Esc]+[Ctrl]+[F]:
Das nächste Zeichen, das dem Zeichen unter dem Cursor entspricht, wird vorwärts gesucht.

10.9 Das Extras-Menü

Set-arg [Ctrl]+[U]:
Viele der Menübefehle akzeptieren eine ganze Zahl als Argument für ihre Ausführung. So können einzelne Tasten oder Befehle wie »Kill-line«, »Next-page« und so weiter auch mehrmals hintereinander ausgeführt werden, indem man mit »Set-arg« die Anzahl der Wiederholungen festlegt. Zur Abfrage der Zahl wird der Cursor in die Statuszeile gesetzt und erwartet die Eingabe einer positiven oder negativen ganzen Zahl (die Zahlen vom Ziffernblock werden jedoch nicht akzeptiert). Direkt nach der Eingabe der Anzahl muß die Taste oder der Menübefehl eingegeben werden.

Set [Esc]+[s]:
erlaubt verschiedene, auch systemabhängige Parameter des Editors zu setzen.

- Screen wechselt die Anzeigeform zwischen eigenem Screen und Workbenchfenster, in dem der Editor dargestellt wird.

- »Interlace« schaltet den Interlace-Modus ein oder aus, der die doppelte Anzahl Zeilen darstellen kann wie der Hires-Modus der Workbench. (Anstatt 256 Zeilen werden 512 Zeilen dargestellt. Da das hochauflösende Bild aus zwei Halbbildern besteht, die abwechselnd gezeigt werden, sinkt die Bildwiederholfrequenz auf 25 Hz, und das Bild flimmert merklich.)
- »Mode«: Der Benutzer kann zwischen zwei alternativen Editoreinstellungen wählen. Nach der Anwahl dieser Funktion springt der Cursor in die Abfragezeile und ermöglicht die Eingabe von »cmode« oder »wrapmode«. In der Einstellung »cmode« wird der Text am Zeilenende nicht umbrochen. Bei »wrapmode« wird eine Zeile umbrochen, wenn sie länger wird als maximal achtzig Zeichen (der übliche Wert für den rechten Rand). Das letzte Wort, das nicht mehr ganz in die Zeile paßt, wird an den Anfang der nächsten Zeile gesetzt und am Ende der vorherigen Zeile ein [Return] eingefügt.

  Mit einem der Option vorgestellten Minus (–) oder Pluszeichen (+) wird die Option aktiviert oder deaktiviert.
- Left<n>: Left gefolgt von einer Zahl n setzt den linken Rand auf die Spalte n.
- Right<n>: Right gefolgt von einer Zahl n setzt den rechten Rand auf die Spalte n.
- Tab<n>: Tab gefolgt von einer Zahl setzt die Breite der Tabulatoren auf n Zeichen.
- Indent<n>: Gibt an, um wie viele Spalten im »cmode« eine neue Verschachtelungsebene eingerückt werden soll.
- Case: Voreingestellt wird bei MEMACS beim Suchen keine Rücksicht auf die Groß- und Kleinschreibung genommen. Mit Case wird die Anweisung gegeben, diese Unterscheidung zu treffen.
- Backup [ON|OFF|SAVE]: Backup On aktiviert die automatische Sicherungsfunktion. Die alte vorhandene Datei wird in Datei.bak umbenannt und bleibt vorhanden, falls man noch einmal auf die ältere Version zurückgreifen will.

Backup Save veranlaßt MEMACS zu überprüfen, ob schon eine Datei existiert, die den gleichen Namen trägt, wie der Dateiname, der dem Puffer für das Abspeichern zugeordnet ist. Falls eine solche Datei existiert, wird der aktuelle Puffer nicht abgespeichert und eine Meldung ausgegeben.

Backup Off ist voreingestellt. Die Sicherungsfunktionen werden abgeschaltet.

Start-macro [Ctrl]+[X]+[(]:
Nach Auswahl dieser Funktion werden alle folgenden Tastatureingaben und Menübefehle als Befehlssequenz abgespeichert, die sich dann beliebig oft nacheinander ausführen lassen.

Stop-macro [Ctrl]+[X]+[)]:
Schließt die Eingabe eines Macros ab.

Execute-macro [Ctrl]+[X]+[e]:
Wenn mit Start-macro und End-macro ein Macro eingegeben wurde, kann es mit Execute-macro beliebig oft ausgeführt werden.

Set-key [Ctrl]+[X] [Ctrl]+[K]**:**
Die Funktionstasten, die [Help]-Taste und alle Tasten des Zehnerblocks können mit Macros belegt werden. MEMACS fragt zuerst in der Statuszeile ab, welche Taste mit einem Macro belegt werden soll. Nach Eingabe der Taste wird das Macro angezeigt, mit dem momentan diese Taste belegt ist. Ein Macro darf maximal 80 Zeichen lang sein. Befehle, die direkt auf die Ausgabe wirken, müssen mit »Quote-char« als Text eingegeben werden, damit sie in tastengebundenen Macros verwendet werden können. Wegen eines Fehlers im Befehl »Set-mark« muß er bei der Macrodefinition immer mit der Tastenkombination [Esc]- gegeben werden. Die Eingabe von [Return] schließt das Macro ab.

Alle aufgeführten Tasten sind mit Macros vorbelegt:

[F 1]	[Ctrl]A[Ctrl]K[Ctrl]Y[Ctrl]M[Ctrl]Y	Zeile vervielfachen
[F 2]	[Ctrl]X[Ctrl]D	Zeile löschen
[F 3]	[Ctrl]Xe	Tastaturmakro ausführen
[F 4]	[Ctrl]V	Nächster Bildschirm
[F 5]	[Esc]v	Vorheriger Bildschirm
[F 6]	[Ctrl]X2	Fenster teilen
[F 7]	[Ctrl]X1	Ein Fenster
[F 8]	[Ctrl]Z	Nach oben blättern
[F 9]	[Esc]Z	Nach unten blättern
[F 10]	[Ctrl]X[Ctrl]F	Puffer abspeichern und MEMACS beenden
	[Hilfe] [Ctrl]J	Zeile einfügen und einrücken
	[Enter] [Ctrl]J	Zeile einfügen und einrücken

Die Tasten des Zehnerblocks sind mit ihren normalen alphanumerischen Werten belegt.

Reset-keys [Esc]+[k]**:**
Alle mit Macros belegbaren Tasten erhalten ihre Standardeinstellung zurück.

Execute-file [Esc]+[e]**:**
MEMACS lädt eine Datei und versucht sie als Macro von MEMACS-Befehlen auszuführen. In der Statuszeile fragt MEMACS nach dem Namen der Macro-Datei.

Execute-line [Ctrl]+[[[Ctrl]+[] **oder** [Esc] [Esc]**:**
MEMACS erlaubt in der Statuszeile die Eingabe eines Menübefehls durch seinen vollständig ausgeschriebenen Namen, wie er im Pull-down-Menü angegeben ist, mit der Angabe von allen benötigten Parametern.

10.10 MEMACS-Befehle, die nur über Tastaturgeben aktiviert werden können.

[Esc]+[Ctrl]+[D]**:**
Erwartet vom Benutzer die Eingabe einer Tastenfolge und zeigt daraufhin an, ob und, falls vorhanden, welche Funktion an diese Tastenkombination gebunden ist.

[Esc]+[Ctrl]+[B]:
Ein MEMACS-Befehl kann an eine neue Tastenkombination gebunden werden. In der Statuszeile wird die Eingabe des kompletten Namens der Funktion erwartet, gefolgt von der neuen Tastenkombination, an die der Befehl gebunden werden soll.

[Esc]+[Ctrl]+[U]:
Mit dieser Funktion läßt sich die Bindung eines Befehls an eine bestimmte Tastenfolge durch die vorherige Funktion wieder aufheben. Nachdem eine belegte Tastenkombination eingegeben wurde, meldet MEMACS »Key is not bound«.

[Esc]+[Ctrl]+[E]:
Die aktuell eingetippte Befehlsfolge wird in der Statuszeile angezeigt. Damit lassen sich MEMACS-Macro-Dateien beim Erstellen leichter überprüfen und bearbeiten.

[Ctrl]+[B]:
In Verbindung mit »Set-arg« ([Ctrl]+[U]) wird der Cursor um die Anzahl von Zeichen nach vorne gesetzt, die mit »Set-arg« angegeben wurde. [Ctrl]+[B] ohne »Set-arg« bewegt den Cursor um ein Zeichen nach vorn.

[Ctrl]+[D]:
Gleiche Wirkung wie [Del].

[Ctrl]+[F]:
In Verbindung mit »Set-arg« ([Ctrl]+[U]) wird der Cursor um die Anzahl von Zeichen nach vorne gesetzt, die mit »Set-arg« angegeben wurde. [Ctrl]+[F] ohne »Set-arg« bewegt den Cursor um ein Zeichen nach vorn.

[Ctrl]+[H]:
Gleiche Wirkung wie [Backspace].

[Ctrl]+[M]:
Gleiche Wirkung wie [Return].

10.11 Anlegen einer Konfigurationsdatei von MEMACS

Eine Datei mit dem Namen »emacs_pro« wird von MEMACS beim Start im aktuellen Verzeichnis und danach im Verzeichnis »S:EMACS« gesucht. Wird sie nicht gefunden, so startet MEMACS mit den normalen Voreinstellungen. Will der Benutzer vor jedem Arbeiten mit MEMACS jedoch einige Voreinstellungen am Editor vornehmen, so kann er eine Startup-datei emacs_pro für MEMACS anlegen, in die er die gewünschten MEMACS-Befehle einträgt. Nach dem jeweiligen Kontext kann man auch mehrere verschiedene Startdateien anlegen, die benutzt werden, wenn das Verzeichnis, in dem sie abgelegt sind, jeweils das aktuelle Verzeichnis ist.

Anhang Druckertreiber 1.3

Folgende Drucker werden unter Betriebssystemversion 1.3 mit schnellen leistungsfähigen Treibern unterstützt:

Alphacom_AlphaPro_101

Typenraddrucker; druckt nur alphanumerische Zeichen.

Schalterstellung:

```
MODUS A: (bidirektionaler Druck)
MODUS B: (kein Zeilenvorschub nach Zeilenrücklauf)
```

Getestete Schnittstellenplatine: 1011PC (IBM PC)

Brother_HR-15XL

Typenraddrucker; druckt nur alphanumerische Zeichen.

Schalterstellungen:

```
Schalter SW1:
EIN
---   ---------------
AUS   x x x x x x x x
      1 2 3 4 5 6 7 8

Schalter SW2:
EIN
---   ---------------
AUS   x x x x x x x x
      1 2 3 4 5 6 7 8
```

Calcomp_ColorMaster

Thermodrucker; druckt Text und Grafik auch farbig.

```
Druckdichte:          Density:
203 x 200 DPI         1
```

Der Druckertreiber kann zwei Drucker betreiben:

- Für den ColorMaster sollte »Paper Size« auf »Narrow Tractor« gesetzt werden.
- Für den ColorView-5912 sollte »Paper Size« auf »Wide Tractor« gesetzt werden, wenn Papier der Größe 11 mal 17 Zoll verwendet wird.

Es sind keine DIP-Schalter vorhanden.

Calcomp_ColorMaster2

Dieser Treiber entspricht dem Treiber Calcomp_ColorMaster.

Das Drucken von Farbgrafiken geschieht etwa doppelt so schnell wie bei dem anderen Treiber, jedoch braucht der Druck wesentlich mehr RAM-Speicher. Ein Farbdruck einer gesamten 8 mal 10 Zoll (1600 x 2000 Punkte) großen Seite erfordert 1.2 Mbyte, der Druck eines formatfüllenden Grafikbidschirms (1600 x 1149 Punkte) 730 Kbyte Speicher.

Beim ColorView-5912 sind bei einer Druckgröße von 10 mal 16 Zoll (2048 x 3200 Punkte) bis zu 2.5 Mbyte Speicher für einen formatfüllenden Farbdruck erforderlich. Ein normaler Grafikbildschirm mit 2048 x 2155 Punkten im Ausdruck erfordert schon 1.7 Mbyte freien Speicher.

Erfolgt der Ausdruck schwarzweiß, wird ungefähr ein Drittel des Speicherplatzes von entsprechenden Farbdrucken benötigt.

Nur Anwender mit massenhaft Speicher werden diesen Druckertreiber verwenden können.

Canon_PJ-1080A

Farbtintenstrahldrucker für Text und Grafik.

```
Druckdichte:          Density:
83 mal 84 DPI         1
```

Schalterstellungen:

```
EIN
---   -----------
AUS   x x x x x x
      1 2 3 4 5 6
```

CBM_MPS1000 (IBM 5152 kompatibel)

Matrixdrucker, druckt Text und Grafik s/w

```
Druckdichte:          Density:
120 mal  72 DPI       1
120 mal 144 DPI       2
240 mal  72 DPI       3
```

```
120 mal 216 DPI        4
240 mal 144 DPI        5
240 mal 216 DPI        6
```

Schalterstellungen:

```
EIN   x
---   ---------------
AUS     x x x x x x x
      1 2 3 4 5 6 7 8
```

Canon BJ-130 (mit Steuerkapsel 48/XL – kompatibel mit IBM 5152)

```
Schalter 1:
EIN   x x x x x x x x x x
---   --------------------
AUS
      1 2 3 4 5 6 7 8 9 10

Schalter 2:
EIN   x x x x x x x x x x
---   --------------------
AUS
      1 2 3 4 5 6 7 8 9 10
```

Diablo_630

Typenraddrucker; nur Druck von alphanumerischen Zeichen.

Schalterstellungen:

```
Schalter 1:
EIN
---   ---------------
AUS   x x x x x x x x
      1 2 3 4 5 6 7 8

Schalter 2:
EIN
---   ---------------
AUS   x x x x x x x x
      1 2 3 4 5 6 7 8
```

Diablo_Advantage_D25

Typenraddrucker; nur Druck von alphanumerischen Zeichen.

Schalterstellungen:

```
Schalter links:
EIN
---   ---------------
AUS   x x x x x x x x
      1 2 3 4 5 6 7 8
```

Eingestellt für XON/XOFF-Protokoll der seriellen Schnittstelle und Zeilenrücklauf ohne Zeilenvorschub.

```
Schalter rechts:
EIN
---   ---------------
AUS   x x x x x x x x
      1 2 3 4 5 6 7 8
```

Diablo_C-150

Farbtintenstrahldrucker; druckt Text und Grafik.

Der Drucker die Einschränkung, daß der Zeilenvorschub immer vier Punkte beträgt.

```
Druckdichte:          Density:
120 mal 120 DPI       1
```

Wird »Paper Size« in Preferences auf »Wide Tractor« gesetzt, beträgt die maximale Druckbreite 8,5 Zoll, wenn eine breite Papierrolle verwendet wird.

Schalterstellungen:

```
Schalter 1:
EIN       x x x
---   ---------------
AUS   x x       x x x
      1 2 3 4 5 6 7 8
```

EpsonQ (und zur Epson »Q«-Serie kompatible 24-Nadel-Drucker wie Starb NB24-10/15)

Matrixdrucker je nach Modell S/W und Farbe, Text und Grafik.

Treiber für alle Drucker, die kompatibel mit EpsonQ (LQ1500, LQ2500, etc.) sind.

```
Druckdichte:          Density:
 90 mal 180 DPI       1
120 mal 180 DPI       2
180 mal 180 DPI       3
360 mal 180 DPI       4
```

»Paper Size« »Wide Tractor« läßt eine maximale Druckbreite von 13,6 Zoll zu, wenn Drucker mit einer 15 Zoll breiten Walze verwendet werden.

Wird für »Paper Type« »Single« angegeben, werden nur 16 der 24 Nadeln verwendet; »Fanfold« benutzt alle 24 Nadeln. Nur wenn beim Grafikdruck das untere Drittel einer Druckzeile merklich schwächer gedruckt wird, muß der Sechzehn-Nadel-Modus gewählt werden.

EpsonX (CBM MPS 1250 und zur Epson »X«-Serie kompatible 9-Nadel-Drucker)

Matrixdrucker; je nach Modell S/W- und Farbe, Text und Grafik.

Alle Drucker, die kompatibel mit EpsonX (EX / FX / JX / LX / MX / RX, etc.) sind. Die Einstellung »Paper Type« »Single« bewirkt einen Zeilenvorschub der Länge:

(Anzahl der vertikal gedruckten Punkte) – 1/3 Punkt

```
Druckdichte:          Density:
120 mal  72 DPI       1
120 mal 144 DPI       2
240 mal  72 DPI       3
120 mal 216 DPI       4
240 mal 144 DPI       5
240 mal 216 DPI       6
```

»Paper Size« »Wide Tractor« ergibt eine maximale Druckbreite von 13,6 Zoll bei Drukkern mit 15 Zoll breiter Walze.

Schalterstellungen:

Commodore MPS1250

```
Schnittstellenmodul:
EIN   x
---   ---------------
AUS     x x x x x x
      1 2 3 4 5 6 7 8

Schalter im Drucker:
EIN   x
---   ---------------
AUS     x x x x x x x
      1 2 3 4 5 6 7 8
```

Epson-1000

```
Schalter 1:
EIN             x x x
---   ---------------
AUS   x x x x x
      1 2 3 4 5 6 7 8
```

```
Schalter 2:
EIN
---   ---------------
AUS   x x x x x x x x
      1 2 3 4 5 6 7 8
```

EpsonXOld (Zum Epson»X« kompatible ältere Drucker mit 8/9 Nadeln speziell STAR Gemini 10-X)

Sollte Ihr Drucker mit dem Epson»X«-Treiber beim Grafikdruck nicht korrekt arbeiten, hilft hoffentlich dieser Treiber für alle sehr alten mit Epson»X« kompatiblen Druckern (EX/FX/JX/LX/MX/RX, etc.)

```
Druckdichte:            Density:
 60 mal  72 DPI         1
120 mal  72 DPI         2       schneller
120 mal  72 DPI         3
240 mal  72 DPI         4
120 mal  72 DPI         5       für alte Modelle
240 mal  72 DPI         6       STAR Drucker
240 mal  72 DPI         7
```

»Paper Size« »Wide Tractor« ergibt eine maximale Druckbreite von 13,6 Zoll bei Drukkern mit 15 Zoll breiter Walze.

Schalterstellungen:

```
Schalter 1 (intern):
STAR Gemini 10-X:
EIN   x x x x x x x x
---   ---------------
AUS
      1 2 3 4 5 6 7 8

Schalter 2 (extern):
EIN   x x
---   -------
AUS       x x
      1 2 3 4
```

Generic

Für Drucker gedacht, für die kein speziell angepaßter Druckertreiber (Commodore, Drukkerhersteller oder Public Domain) erhältlich ist.

Howtek_Pixelmaster

Kunststoff-Tintenstrahldrucker, S/W- und Farbdruck, druckt Text und Grafik.

```
Druckdichte:          Density:
 80 mal  80 DPI       1
120 mal 120 DPI       2
160 mal 160 DPI       3
240 mal 240 DPI       4
```

Maximale bedruckbare Fläche 8 x 10 Zoll.

keine DIP-Schalter

HP_DeskJet

Tintenstrahldrucker, S/W Text und Grafik.

```
Druckdichte:          Density:
 75 mal  75 DPI       1
100 mal 100 DPI       2
150 mal 150 DPI       3
300 mal 300 DPI       4
```

Maximale bedruckbare Fläche 8 x 10 Zoll.

Schalterstellungen:

```
Schalter 1:
EIN
---   ---------------
AUS   x x x x x x x x
      1 2 3 4 5 6 7 8

Schalter 2:
EIN
---   ---------------
AUS   x x x x x x x x
      1 2 3 4 5 6 7 8
```

HP_LaserJet (und mit LaserJet PLUS/LaserJet II kompatible Drucker)

Laserdrucker, S/W Text und Grafik.

```
Druckdichte:          Density:
 75 mal  75 DPI       1
100 mal 100 DPI       2
150 mal 150 DPI       3
300 mal 300 DPI       4
```

Maximale bedruckbare Fläche 8 x 10 Zoll.

Keine DIP-Schalter vorhanden.

HP_PaintJet

Farbtintenstrahldrucker Text und Grafik.

```
Druckdichte:            Density:
180 mal 180 DPI         1
```

Schalterstellungen:

```
EIN
---   -------------
AUS   x x x x x x x
      1 2 3 4 5 6 7
```

HP_ThinkJet

Tintenstrahldrucker S/W Text und Grafik.

```
Druckdichte:            Density:
 96 mal  96 DPI         1
192 mal  96 DPI         2
```

Schalterstellungen:

```
EIN     x       x
---   ---------------
AUS   x   x x x   x x
      1 2 3 4 5 6 7 8
```

Imagewriter II

Matrixdrucker; S/W und Farbe, Text und Grafik.

```
Druckdichte:            Density:
 80 mal  72 DPI         1
120 mal  72 DPI         2
144 mal  72 DPI         3
160 mal  72 DPI         4
120 mal 144 DPI         5       zwei Durchgänge
144 mal 144 DPI         6       zwei Durchgänge
160 mal 144 DPI         7       zwei Durchgänge
```

Schalterstellungen:

```
Imagewriter:
Schalter 1:
EIN   x x x x     x x
---   ---------------
AUS           x x
      1 2 3 4 5 6 7 8

Schalter 2:
EIN         x
---   -------
AUS   x x x
      1 2 3 4

Imagewriter II:
Schalter 1:
EIN   x x x x x   x x
---   ---------------
AUS             x
      1 2 3 4 5 6 7 8

Schalter 2:
EIN       x x
---   -----------
AUS   x x     x x
      1 2 3 4 5 6
```

NEC_Pinwriter (P5/P6/P7/P9/P2200 kompatible Drucker)

Matrixdrucker, S/W Text und Grafik.

Treiber für alle mit NEC Pinwriter kompatible 24-Nadel-Drucker

```
Druckdichte:            Density:
 90 mal 180 DPI         1
120 mal 180 DPI         2
180 mal 180 DPI         3
120 mal 360 DPI         4       zwei Durchgänge
180 mal 360 DPI         5       zwei Durchgänge
360 mal 180 DPI         6       zwei Durchgänge
360 mal 360 DPI         7       zwei Durchgänge
```

Bei »Paper Size« »Wide Tractor« ist eine maximale Druckbreite von 13,6 Zoll bei Druckern mit 15 Zoll Walzenbreite.

Schalterstellungen:

```
P9XL:
Schalter 1:
EIN
---   ---------------
AUS   x x x x x x x x
      1 2 3 4 5 6 7 8

Schalter 2:
EIN           x
---   ---------------
AUS   x x x x   x x x
      1 2 3 4 5 6 7 8

Schalter 3:
EIN               x
---   ---------------
AUS   x x x x x x   x
      1 2 3 4 5 6 7 8
```

Okidata_92

Matrixdrucker; S/W Text und Grafik.

Zeilenvorschub im Grafikmodus immer 7/72 Zoll.

```
Druckdichte:          Density:
72 mal  72 DPI        1
```

Okidata_293I

Matrixdrucker; S/W und Farbe, Text und Grafik.

Treiber für Typ 292 und 293 unter Verwendung des IBM-Schnittstellenmoduls.

Für »Paper Type« »Single« ist der Zeilenvorschub:

(Anzahl der vertikal gedruckten Punkte) – 1/2 Punkt

```
Druckdichte:          Density:
120 mal 144 DPI       1
240 mal 144 DPI       2
120 mal 288 DPI       3       zwei Durchgänge
240 mal 288 DPI       4       zwei Durchgänge
```

Bei »Paper Size« »Wide Tractor« beträgt die größte mögliche Druckfläche 13,6 Zoll, wenn ein 15-Zoll-Traktor verwendet wird.

Folgende Jumper sind zu setzen:

```
Schnittstelle: ML-292/293 Personality Modul
SP1
        *---*           Jumper setzen

SP4     *   *---*
        *---*   *
```

Okimate-20

Thermodrucker; S/W Farbe, Text und Grafik.

Der Zeilenvorschub ist immer eine gerade Anzahl von Punkten.

```
Druckdichte:            Density:
120 mal 144 DPI         1

parallele Schnittstelle
EIN           x
---   -----------
AUS   x x x x   x
      1 2 3 4 5 6
```

Achtung! Der Schalter 5 steuert bei einigen Modellen den Abstand zwischen den Grafikzeilen.

```
serielle Schnittstelle
Schalter 1:
EIN   x x       x
---   -----------
AUS       x x x
      1 2 3 4 5 6

Schalter 2:
EIN           x
---   -----------
AUS   x x x x   x
      1 2 3 4 5 6
```

Achtung! Die Einstellung von Schalter 1 legt Baudrate 9600, das XON/XOFF-Protokoll, eine Wortlänge von 8 Bit und kein Paritybit fest.

Der Schalter 5 steuert bei einigen Modellen den Abstand zwischen den Grafikzeilen.

Quadram_QuadJet

Farbtintenstrahldrucker; Text und Grafik.

```
Druckdichte:            Density:
83 mal  84 DPI          1
```

Es sind keine DIP-Schalter vorhanden.

Qume_LetterPro_20

Typenraddrucker, druckt nur Alphazeichen.

Schalterstellungen:

```
Schalter A:
EIN              x
---   ---------------
AUS   x x x x x x   x
      1 2 3 4 5 6 7 8

Schalter B:
EIN    x x
---  ---------
AUS  x     x x
     1 2 3 4 5
```

Toshiba_P351C

Matrixdrucker (24 Nadeln); S/W und Farbe, Text und Grafik.

```
Druckdichte:            Density:
180 mal 180 DPI         1
360 mal 180 DPI         2
```

»Paper Size« »Wide Tractor« ermöglicht eine maximale Druckbreite von 13,5 Zoll bei Verwendung einer 15-Zoll-Druckwalze.

Schalterstellungen:

```
Schalter 1:
EIN       x x   x x
---   ---------------
AUS   x x     x     x
      8 7 6 5 4 3 2 1

Schalter 2:
EIN            x
---   ---------------
AUS   x x x x x   x x
      8 7 6 5 4 3 2 1
```

```
Schalter 3:
EIN
---   ---------------
AUS   x x x x x x x x
      8 7 6 5 4 3 2 1

Schalter 4:
EIN   x x
---   ---------------
AUS       x x x x x x
      8 7 6 5 4 3 2 1
```

Toshiba_P351SX (321SL, 321SLC, 341SL)

Matrixdrucker; S/W und Farbe, Text und Grafik.

```
Druckdichte:          Density:
180 mal 180 DPI       1
360 mal 180 DPI       2
180 mal 360 DPI       3       zwei Durchgänge
360 mal 360 DPI       4       zwei Durchgänge
```

Bei »Paper Size« »Wide Tractor« ist eine maximale Druckbreite von 13,5 Zoll bei Drukkern mit 15 Zoll Walzenbreite möglich.

Jumper-Konfiguration:

```
1---2, 8---9, 7---10, 11---12, 13---14
```

Xerox_4020

Farbtintenstrahldrucker; Text und Grafik.

Der Zeilenvorschub ist immer 4 Punkte.

Der Treiber ist identisch mit dem Diablo_C-150 Treiber mit dem Unterschied, daß zur Unterstützung der besonderen Fähigkeit des Xerox-Druckers alle schwarzen Punkte zweimal ausgegeben werden, um kräftigere schwarze Flächen zu erzeugen.

```
Druckdichte:          Density:
121 mal 120 DPI       1
242 mal 240 DPI       2
```

Da der Drucker eine Auflösung von 240 DPI nicht unterstützt, wird bei der Einstellung 2 eine Zeile zweimal um 1/240 Zoll versetzt ausgedruckt.

Dadurch wird eine wesentlich höhere Farbsättigung erreicht. Das hat allerdings den Nachteil, daß bei großen zusammenhängenden Farbflächen das Papier zu sehr mit Farbe getränkt wird.

Wird »Paper Size« »Wide Tractor« gewählt, beträgt die maximale Druckbreite 9,0 Zoll bei Druckern mit breiter Transportwalze.

Schalterstellungen:

```
OFF  |ON    OFF  |ON
AUS  |EIN   AUS  |EIN
1 x  |      1 x  |
2 x  |      2    |x
3 x  |      3 x  |
4 x  |      4 x  |
            5 x  |
            6 x  |
            7 x  |

Links       RECHTS
```

Programmmierer-Buch

1 Einleitung

Der Amiga war der erste Computer, der Multitasking auf Personal Computern möglich machte. Heute zieht nicht zuletzt IBM mit diesem wichtigen Leistungsmerkmal unter OS/2 nach.

Multitasking ist die Fähigkeit, mehrere verschiedene Programme quasi gleichzeitig abzuarbeiten, indem jedes Programm einen Teil der Rechenzeit zugeteilt bekommt und danach auf ein wartendes Programm umgeschaltet wird. Das geschieht in der Regel so schnell, daß der Anwender glaubt, alle Programme würden gleichzeitig laufen.

Die Systemsoftware des Amiga hat ein konsequent modulares Konzept und kapselt low-level Gerätetreiber (oder Treiber für die Spezialchips) von der darüberliegenden Schicht von Handlern oder Dateisystemen. Die für den Programmierer nutzbaren Funktionen des Betriebssystems sind in verschiedenen Bibliotheken thematisch sortiert organisiert, die keine festen Einsprungadressen haben, sondern über Sprungtabellen erreichbar sind. Die Sprungtabellen mit den Offsets dafür liegen in Form der ».fd«-Dateien für Basic-Programmierer vor, oder der Linklibrary »amiga.lib«, die vom Linker zum Programm dazugebunden wird. Direkteinsprünge über konstante Adressen können möglicherweise manchmal funktionieren, werden aber garantiert bei einer neuen Betriebssystemversion zum Absturz des Programmes führen. Verlassen Sie sich auf keine konstante Adresse beim Amiga-Betriebssystem, außer der Speicheradresse vier. Hier wird der Einsprungpunkt für alle Betriebssystemroutinen gespeichert.

Ein Hochsprachen-Programmierer merkt von dem Zugriff über Sprungtabellen jedoch nichts, da er nach dem Öffnen einer Bibliothek mit der Funktion »OpenLibrary()« auf alle Funktionen dieser Bibliothek direkt zugreifen kann. Der Linker setzt mit Hilfe der Linklibrary amiga.lib alle Sprungadressen der aufgerufenen Funktionen korrekt relativ zum Basispointer auf die Bibliothek, der von OpenLibrary() zurückgeliefert wird. Lattice C bietet seit der Version 4.0 die Möglichkeit, über Prototypdateien, in denen die Offsets für den Einsprung in die Bibliothek gespeichert sind, Bibliotheksaufrufe ohne Linken an die amiga.lib auszuführen.
Bei der Programmierung in Assembler ist das Aufrufen von Bibliotheksfunktionen etwas aufwendiger. Die Basisaddresse der jeweiligen Bibliothek muß in das Register A6 geladen werden und danach ein indirekter Einsprung mit dem korrekten negativen Offset in die Bibliothek erfolgen. Diese Offsets sind in der Systembibliothek des Amiga unter dem Namen _LVO»name« gespeichert. Der Aufruf der Funktion kann also mit JSR _LVO-»name«(A6) erfolgen, nachdem A6 mit dem korrekten Wert geladen wurde, den OpenLibrary() zurückgeliefert hat. Bei der DOS.library könnte man sich als Programmierer die

Besonderheit zunutze machen, daß alle bisherigen Versionen nicht unbedingt über das Basisregister A6 angesprungen werden müssen, sondern die Möglichkeit besteht, jedes beliebige Register zu benutzen, um mit einem JSR _LVO»name«(An) die DOS-library zu öffnen. Diese Technik funktioniert im Moment zwar, aus Kompatibilitätsgründen sollte man bei der Erstellung von Software darauf verzichten und auch für den Aufruf der DOS.library A6 als Basisregister laden.

Einige Register des Prozessors werden vom Betriebssystem genutzt und sollten daher vom Anwendungsprogrammierer nicht benutzt werden. Bibliotheksfunktionen überschreiben meist die Register D1, A0 und A1. In D0 wird das Ergebnis zurückgegeben, das die Funktion liefert.

Alle Prozesse, die unter Amiga-DOS laufen, teilen sich den Hauptspeicher. Leider ist es dadurch nicht möglich, den Speicherbereich eines Programmes vor dem Überschreiben durch ein weiteres Programm zu schützen. Ebenso unterstützt das Betriebssystem kein Resource Tracking, führt also nicht Buch über Speicher, geöffnete Bibliotheken und Gerätetreiber oder Zugriff auf die Hardware, um dem Programmierer den Abbruch oder das Beenden des Programms zu erleichtern. Der Programmierer ist für die Buchführung und das korrekte Schließen aller Systemresourcen verantwortlich und muß ebenso Zugriffe auf Speicherbereiche fremder Prozesse vermeiden, da er mit diesen Daten in der Regel nichts anfangen kann und das Betriebssystem höchstens zum Abstürzen bringt (Kranke destruktive Hirne schwelgen bestimmt schon in Gedanken, was sie alles vermurksen können. Die meisten Anwender und Kunden können jedoch auf solchen Schwachsinn verzichten).

Programmierer, die das Arbeiten auf Großrechnern oder UNIX-Maschinen gewohnt sind, werden jetzt vielleicht mitleidig lächeln und sagen, daß die Schutzmechanismen des Amiga-DOS lächerlich (weil nicht vorhanden) sind. Sie sollten sich aber klarmachen, daß der Amiga keine Maschine ist, die von mehreren verschiedenen Personen gleichzeitig genutzt wird und somit Abstürze allein in der Verantwortung des Benutzers (Programmierers) und seiner schlecht programmierten Programme liegen. Programme, die sauber alle Resourcen handhaben, werden nicht abstürzen, es sei denn, sie werden im wahrsten Sinne des Wortes von einem fehlerhaften Programm torpediert. Eine Memory Management Unit, die einen wirksamen Schutz gegen Speichersegmentverletzungen bietet, wenn das Betriebssystem sie ausnutzt, da sie die Verwaltung des gesamten Speichers übernimmt und Überschreiber nicht zuläßt, ist in einem normalen Amiga nicht standardmäßig vorhanden.

1.1 Eine Übersicht über die Systemsoftware

Für den Anwender gibt es zwei Schnittstellen: Die Workbench ist eine grafische, mausgesteuerte Schnittstelle, um das Dateisystem zu repräsentieren und das Starten von Programmen zu erlauben. Die SHELL ist eine alphanumerische Eingabe, die weitergehende Kontrolle über das System wie die Workbench erlaubt. Sie benutzt den Console-Handler CON: oder NEWCON:.

Das Input device, eine Softwareschnittstelle, die alle Tastatur- und Mauseingaben des Benutzers aufarbeitet, organisiert den Datenstrom, den das Keyboard device (Tastaturtreiber), das Gameport device (Mausanschlüsse), das Timer device (präzise Systemuhr) liefern, und verschickt sie an den Empfänger.

Die grafische Darstellung von Programmen in Fenstern und das Aufbereiten von Daten aus dem Input device übernimmt Intuition, das für überlappende Fenster auf die Layers.library zurückgreift und zum Zeichnen die Grafics.library benutzt.

Das Timesharing, die Speicherverwaltung und die Prozeßkommunikation werden vom Betriebsystemkern EXEC verwaltet.

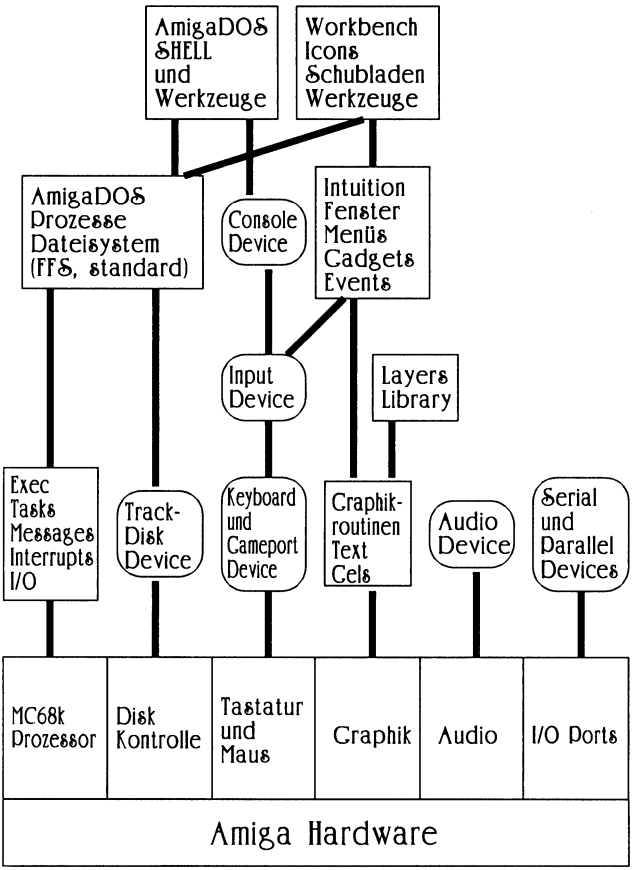


Bild 1.1 Übersicht über die Systemsoftware

Das Amiga-DOS übernimmt die Verwaltung von Daten im Dateisystem, ist also für das Anlegen des Verzeichnisbaumes und der einzelnen Dateien zuständig. Alle Daten werden als Datenstrom präsentiert und erlauben daher auch das Ansprechen von Druckern oder der Sprachausgabe als Teil des Dateisystems. Das DOS benutzt für jedes Gerät einen Handler-Prozeß, der mit Hilfe eines spezifischen Gerätetreibers die Daten auf dem Gerät verwaltet, und bietet nach außen für den Benutzer eine einheitliche, bedienerfreundliche Schnittstelle.

Im Gegensatz zu dem gesamten Rest des Betriebssystem wurde das DOS nicht in C geschrieben, sondern ist eine Portierung des in BCPL geschriebenen Betriebssystems TRIPOS, einem als Konkurrenz zu UNIX gedachten Produkt, das UNIX jedoch nicht im geringsten das Wasser reichen kann. BCPL hat einige Eigenheiten. So kennt es nur einen einzigen Datentyp Wort, der immer ab einer Langwortadresse (ein Langwort ist 32 Bit lang) gespeichert wird. Zeiger auf diesen Datentyp zeigen immer nur auf ein vollständiges Wort. Um die korrekte C-Adresse im Speicher zu erhalten, muß der Pointer mit vier multipliziert (oder zweimal nach links geshiftet) werden, da C jedes Byte im Speicher adressiert. Ebenfalls werden Zeichenketten im Speicher anders als in C abgelegt. Im ersten Byte steht die Länge des Strings, die maximal 80 Zeichen nicht übersteigen darf, danach folgt die Zeichenkette. In C wird die Länge des Strings nicht gespeichert, sondern das Ende der Zeichenkette wird durch ein Byte mit dem Wert 0 gekennzeichnet. Zu allem Überfluß verwaltet BCPL den Stack von der Basisadresse aufsteigend, entgegengesetzt dem normalen Verhalten des MC68k Prozessors, der den Stack absteigend verwaltet.

Die Kenntnis dieser Tatsachen ist nicht wichtig, um selbst BCPL programmieren zu können, sondern um das in BCPL geschriebene DOS richtig benutzen zu können.

1.2 Eine Übersicht über die Hardware des Amiga

Allen an der Hardware interessierten Programmierern soll nun ein kleiner Überblick über das Hardwarekonzept des Amiga geboten werden.

1.2.1 Der MC68k

Als CPU besitzt der Amiga einen Motorola MC68000 Prozessor mit einer Taktfrequenz von 7,3 MHz. Dieser Prozessor kann den Hauptspeicher über einen 24-Bit-Adreßbus linear adressieren, das heißt, es stehen bis zu 16 Mbyte Speicher zur Verfügung, von denen das Betriebssystem bis zu 9,5 Mbyte benutzen kann. Zur Kompatibilität mit den größeren Prozessoren der MC68k-Familie müssen Zeiger (Adressen) immer 32 Bit lang sein. Der Datenbus des 68000 ist nur sechzehn Bit breit, die internen Register des Prozessors fassen allerdings 32 Bit breite Worte. Der Amiga kann mit Hilfe einer Turbokarte nämlich auch auf die Verwendung eines 68020 oder 68030 Prozessors umgerüstet werden, der eine vielfache Verarbeitungsgeschwindigkeit des 68000 hat. Diese Prozessoren sind echte 32-Bit-

Prozessoren, deren Adreß- und Datenbus 32 Bit breit ist. Außerdem kann optional auch ein Fließkommaarithmetikprozessor benutzt werden, der numerische Berechnungen wesentlich beschleunigt.

1.2.2 Die Customchips

Paula, Agnus und Denise sind drei Spezialchips (Customchips) des Amiga für Grafik und Sound. Sie können auf einen speziellen Speicherbereich, das sogenannte CHIP-RAM in Konkurrenz zum Prozessor per Direktzugriff (DMA) auf den Speicher zugreifen, wenn es der Prozessor gerade nicht tut. Sie besitzen aber für den Fall Zugriffspriorität vor dem Prozessor, wenn aufwendige Grafik darzustellen ist, und bremsen deshalb den Zugriff des Prozessors auf diesen Speicherbereich. Der Prozessor kann ungestört in voller Geschwindigkeit auf den Rest des Speichers zugreifen (siehe Kapitel zwei FASTMEMFIRST).

Die Customchips kontrollieren die Funktion des 68000, indem sie das DTACK-Signal erzeugen, das eine erfolgreiche Datenübertragung signalisiert, und die Interrupts auswerten. Der Chip Paula steuert die Interruptverarbeitung. Es gibt sechzehn physikalische Quellen für einen Interrupt, die Paula entscheidet ob, und mit welcher Priorität der Prozessor unterbrochen werden soll:

- zwei von externer Hardware
- der Copperinterrupt, wenn der Videostrahl eine bestimmte Position erreicht hat
- die vier Audiokanäle melden, wenn ein Audioblock abgearbeitet ist
- Blitter mit der Arbeit fertig
- Disketteninterrupt Synchronmarkierung gefunden
- Disketteninterrupt Diskettenblock geladen
- 8250-Interrupt-Timer
- 8250-Interrupt-Tastatur
- Übertragungspuffer der seriellen Schnittstelle leer
- Empfangspuffer der seriellen Schnittstelle voll

Der Amiga besitzt fünfundzwanzig DMA-Kanäle, die an bestimmte Aufgaben gebunden sind und nicht für andere Aufgaben genutzt werden können:

- Zugriff auf Bitplanes zum Aufbau von Screens (6)
- Steuerung der Hardwaresprites (8)
- Zugriff des Coppers auf seine Copperliste (1)
- Verschieben von Speicherbereichen durch den Blitter (4)
- DMA zur Steuerung der Diskettenlaufwerke (2)
- Vier Audiokanäle (4)

Der Bildschirmaufbau wird durch die Überlagerung von Playfields organisiert (mehrere Intuition-Screens), indem mehrere zusammengehörige Bitebenen als Inhalt eines Playfields interpretiert werden. Bits mit jeweils dem gleichen Abstand zum Anfang der jeweiligen Bitebene werden als Adresse des Farbregisters des Coppers interpretiert, und der entsprechende Punkt auf dem Schirm in der in diesem Register eingetragenen Farbe eingefärbt.

Außerdem können die Customchips bis zu 8 Hardwaresprites auf dem Bildschirm unabhängig von den Playfields auf dem Schirm darstellen. Die Sprites können 16 Pixels breit und 256 Pixels lang sein. Um größere Objekte zu erzeugen, können mehrere Sprites zu einem Objekt zusammengefaßt werden. Ihre Anzahl kann mit Tricks vergrößert werden, die für die VSprites der Grafics-library angewendet werden.

Der Copper ist ein Mikroprozessor, der in seinem Arbeitstakt mit dem Videostrahl synchronisiert ist. Er besitzt mehrere Maschinensprachebefehle, mit denen er auf bestimmte Positionen des Videostrahls warten kann, um dann verschiedene Hardwareregister mit neuen Werten zu laden. Zum Beispiel kann er auf eine bestimmte Zeile auf dem Bildschirm warten, und dann Zeiger auf Playfields umladen, um zwei vertikal überlappte Screens darzustellen. Er kann auch einen CPU-Interrupt auslösen, um bestimmte Aktionen der CPU zu starten, oder den Blitter Daten schaufeln lassen, um Animationen auf dem Bildschirm darzustellen.

Der Blitter ist ein Knecht, der Daten sehr schnell im CHIP-RAM hin- und herkopieren kann. Dabei kann er drei verschiedene Eingabekanäle zu einem Ausgabekanal durch Links-, Rechtsschieben oder 256 logische Verknüpfungsoperationen vereinen. Weiterhin kann er im CHIP-RAM Linien ziehen oder Flächen füllen.

Die Paula steuert vier Audiokanäle, die separate Eingabespeicher besitzten, die als digitale Wellenformen interpretiert werden und als Ton ausgegeben werden.

Das Laden von Daten von Diskette geschieht ebenfalls per DMA. Ein gesammter Sektor wird am Stück in den Speicher des Amiga übertragen und vom Blitter dekodiert, ohne daß sich der 68000 Chip mit dieser Aufgabe beschäftigen müßte. Ungeschickte Algorithmen für die Auswertung dieser Daten von Amiga-DOS führen allerdings zu der teils fürchterlich langsamen Geschwindigkeit von Amiga-DOS, wenn Inhaltsverzeichnisse geladen werden (das ändert sich hoffentlich auch noch im Laufe der kommenden Betriebssystemversionen).

1.2.3 Die Trennung zwischen CHIP- und FAST-RAM

Nur die unteren 512 Kbyte Speicher (mit der neuesten Version des Chip Fat-Agnus 1 Mbyte Speicher) stehen den Customchips zur Verfügung. Den übrigen Speicher (das FAST-RAM) trennt die Architektur des Amiga von den Customchips und läßt nur den Prozessor beim Zugriff auf das FAST-RAM passieren.

1.2.4 Der Kickstart

Der Speicherbereich für das Kickstart-ROM (beziehungsweise das RAM beim Amiga 1000, in das das Betriebssystem von Diskette geladen wird) ist nur für den Prozessor erreichbar und liegt außerhalb des FAST- oder CHIP-RAM-Speichers. Hier befinden sich die meisten Bibliotheken des Betriebssystems, einige Bibliotheken fanden im ROM jedoch keinen Platz mehr und werden von Diskette oder Platte nachgeladen. Die aktuelle Version ist Kickstart 34.5, was man durch den Befehl VERSION auf der SHELL abfragen kann. Die Version 34.5 unterscheidet sich von der Version 33.180 nur an zwei Punkten. Zum einen erscheint bei der Bootaufforderung zum Einlegen der Workbench die Versions-

nummer 1.3 anstatt 1.2, zum anderen kann die Version 34.5 von autobootfähigen Erweiterungskarten oder der Recoverable-RAM-Disk booten. Durch die neu eingefügten Autoboot-Routinen wurden einige Bibliotheksfunktionen um einige Bytes verschoben, so daß ihre Einsprungsadresse nicht mehr mit der Version 1.2 des Kickstart übereinstimmt. (Direktsprünge ins Kickstart fallen deswegen auf die Nase, unter Kickstart 1.4 würden sie sowieso nicht mehr funktionieren.)

1.2.5 Die Timerbausteine

Der Amiga besitzt auch zwei 8520 Peripheriebausteine, die die parallele und die serielle Schnittstelle steuern, verschiedene Timer enthalten und von dem Timer.device benutzt werden. Beide Bausteine sind voll ausgelastet und können nicht direkt in Maschinensprache genutzt werden.

2 Programmierrichtlinien für das Programmieren in C

Wer mehr als kleine Prograämmchen und Routinchen in der Sprache C realisieren will, merkt schnell, daß es nicht nur absolut notwendig ist, ein sauberes und detailliertes Konzept vor der Realisierungsphase zu erarbeiten, sondern es ist auch sinnvoll beim Kodieren der einzelnen Module des Programmpakets gewisse Regeln für den Programmierstil einzuhalten.
Bei der Arbeit in Projektgruppen ist das Befolgen dieser Regeln von allen Mitarbeitern überlebensnotwendig, um es allen Mitwirkenden zu erleichtern, sich auch in fremde Programmteile einzuarbeiten, was für die spätere Wartung und Entfernung von Programmierfehlern wichtig ist. Viele Softwarehäuser schreiben Programmierrichtlinien sogar bindend fest.

2.1 Formatierung von C-Programmen

Ein C-Programm sollte von einem Modulkopf eingeleitet werden, in dem der Name des Moduls, der Name des Autors, das Erstellungsdatum und Änderungen mit Datum und eventuell Namen des Programmierers dokumentiert werden. Meist ist es auch sehr nützlich, die Funktion des Moduls im Modulkopf zu beschreiben, auch wenn parallel zur Kodierung die Dokumentation geschrieben wird.

```
/*************************************************************/
/* Modul:          pushval                                   */
/*-----------------------------------------------------------*/
/* Autor:          Wilfried Häring                           */
/* Datum:          1.4.1989 (April, April)                   */
/* Aenderungen:    2.4.1989 komplett weggeschmissen und neu  */
/*                          programmiert                     */
/*                 3.4.1989 7 Fehler korrigiert und 17 neu   */
/*                          gemacht                          */
/*                 4.4.1989 Ideen geklaut, die aber nicht    */
/*                          brauchbar sind                   */
/*                 5.4.1989 Erstmals weniger als 10 Kbyte    */
/*                          Warnings                         */
```

```
/*                  6.4.1989 Modul wird nicht mehr gebraucht,   */
/*                           Planung komplett ueber den Haufen  */
/*                           geworfen                           */
/*                  7.4.1989 Neue Planung ueber den Haufen      */
/*                           geworfen, alte Planung wieder in   */
/*                           Kraft gesetzt                      */
/*                  ...                                         */
/*                  nächtes Jahr Weihnachten                    */
/*                           erste Probelaeufe mit Endkunden    */
/*                           als Betatester                     */
/*-------------------------------------------------------------*/
/* Beschreibung:                                                */
/*                  Ich finde es absolut super, daß Sie diesen  */
/*                  Schwachsinn bis jetzt gelesen haben. Das    */
/*                  obige Negativbeispiel von perfekter und     */
/*                  durchaus üblicher Projektplanung kann Sie   */
/*                  hoffentlich auf den Pfad der Programmier-   */
/*                  tugend führen.                              */
/***************************************************************/
```

Kommentare sind unabdingbar und sollten häufig in Programmtexte hineingeschrieben werden. Es gibt kaum jemand, der sich beschwert, ein Programmtext sei zu geschwätzig gewesen, vielmehr sieht man oft Programme, in dem die einzige Kommentarzeile den abgekürzten Name des Programms enthält. Nicht immer ist C eine selbstdokumentierende Sprache...

```
/* Ein Zeilenkommentar wird eine Spalte weiter     */
/* eingerückt als die Statements und wird in jeder */
/* Zeile neu begonnen.                             */
```

Nach einem Statement kann ein kurzer Kommentar angefügt werden:

```
if ( condition )                /* kurze Bemerkung */
{
...
}  /* if ( condition ) */
/* Kennzeichen, welche logische Ebene abgeschlossen wurde */

typedef struct
{
   char name[20];            /* Nachname            */
   char prename[20];         /* darf nicht leer sein */
   char birthdate[8];        /* Geburtsdatum        */
                             /* in der Form dd.mm.yy */
} Person;
```

Die Namen für Funktionen, Macros, Typen und Variablen sollten die jeweilige Verwendung an der Schreibweise erkennbar machen.

Prozeduren sollten in Kleinbuchstaben geschriebene Namen besitzen. Namen, die außerhalb der Quelldatei in anderen Quelldateien verwendet werden sollen, werden mit einer vorgestellten Abkürzung des Modulnamens gekennzeichnet. Dies widerspricht der Namensgebungsphilosophie des Amiga-Betriebssystems deutlich und ermöglicht daher eine klare Unterscheidung zwischen den Bibliotheksfunktionen und selbst implementierten Funktionen.

Funktionen:

```
pshv_add, pshv_sub, transformiere_weltkoordinaten, ...
```

Wenn der Lattice-C-Compiler verwendet wird, sollte die Parameterliste bei der Funktionsdefinition mit Typendeklaration angegeben werden, wie es der ANSI-Standard vorschlägt (Der Aztec-C-Compiler unterstützt die ANSI-Richtlinien noch nicht):

```
pshv_add ( int a, int b, struct Screen *screen_p )
{
 ...
}
```

Komplexere Datentypen sollten für eine einfachere Verwendung mit typedef definiert werden und Erleichtern das Lesen von Programmen erheblich. Der Name sollte mit einem Großbuchstaben beginnen, und wenn er aus dem Modul exportiert werden soll, mit einer Abkürzung des Modulnamens gekennzeichnet werden.

Typen:

```
typedef struct pshv_alles_drin
{
   struct pshv_alles_drin        *next;       /* einfache Liste */
   int                           intarray [10][10][10];
   ULONG                         flags;
} Pshv_alles_drin;
```

Macro- oder Konstantennamen, die mit #define vereinbart werden, sollten in Großbuchstaben benannt werden, damit sofort optisch klar wird, daß es sich um eine Macroersetzung handelt, die vom C-Präprozessor ausgeführt wird.

Macros:

```
#define PSHV_MAX ((a) > (b) ? (a) : (b))
```

Variablen sollten mit Kleinbuchstaben bezeichnet werden. Es ist sinnvoll, Zeigervariablen mit einem nachgestellten p für pointer und Zeichenketten(vektoren) mit nachgestellten s oder str zu kennzeichnen.

Variablen:

```
int     pshv_i;
char    *pshv_buf_str;
int     *pshv_reference_p;
```

Blöcke fassen Statements zu logischen und funktionellen Untereinheiten zusammen. In jedem geöffneten Block können automatische Variablen angelegt werden, die nach Verlassen des Blocks wieder freigegeben werden und somit ihre Gültigkeit wieder verlieren. Die öffnende Klammer steht in der gleichen Spalte wie das übergeordnete Statement allein in einer Zeile, Variablendefinitionen fangen in der gleichen Spalte darunter an. Die Statements werden um einen Tab oder weniger, zumindest um ein Leerzeichen eingerückt. Die schließende Klammer steht in der gleichen Spalte wie die öffnende Klammer und wird höchstens von einem Kommentar gefolgt.

```
{
Variablen;
  statements;
  ...
  statements;
}
```

Eine Ausnahme dieser Regel bildet die »do...while«-Schleife, bei ihr folgt auf die schließende Klammer der logische Ausdruck.

```
do
{
  statements;
  ...
  statements;

} while ( condition );
```

Ebenfalls folgt die switch-case-Konstruktion einer etwas veränderten Regel. Die einzelnen Auswahlen case ... : beginnen in der gleichen Spalte wie das übergeordnete switch.

```
switch ( variable )
{
case EINS :
  statements;
  break;
case ZWEI :
  statements;
  /* nobreak */
default:
  statements;
  break;
}
```

Werden bei logischen Ausdrücken, for-Schleifen oder Funktionsköpfen die Parameterlisten oder Ausdrücke zu lang, sollten sie entweder am Trennungszeichen zwischen den einzelnen Parametern oder am Verknüpfungszeichen mit der niedrigsten Priorität getrennt werden.

Statements:

```
for ( Initialisierung; Abbruchkriterium; Fortschaltung )
{
  statements;
  ...
  statements;
}
```

oder

```
for ( Initialisierung aller Variablen;
      Abbruchkriterium;
      Schleifenfortschaltung )
{
  statements;
  ...
  statements;
}

foo ( parameter1, parameter2, parameter3,
      parameter4 );

if ( ( x = foo1 ( harri ) ) ||
     ( x = foo2 ( harri ) ) )
{
  statements;
  ...
  statements;
}
else if ( ... )
else if ( ... )
```

2.2 Einpassen in das Betriebssystem

Das Betriebssystem verlangt von binären, ausführbaren Programmen auch einige Grundregeln, die meist nur Assemblerprogrammierer explizit einhalten müssen.

Systemweit bekannte und nutzbare Datenstrukturen müssen im Datenformat und den initialisierten Werten gewisse Regeln befolgen:

- Alle Datentypen außer UBYTE (unsigned char) und BYTE (char) müssen auf geraden Wortaddressen beginnen. Dadurch kann es nötig werden, daß in Strukturen »Löcher« mit leeren Bytes gestopft werden müssen.
- Alle Adreßzeiger sind 32 Bit lang.
- Alle Datenfelder (auch Zeiger) einer Struktur, die keinen bestimmten Wert enthalten sollen, müssen explizit auf NULL (0) initialisiert werden.
- Alle reservierten Felder müssen zur Kompatibilität mit zukünftigen Betriebssystemversionen auf NULL (0) initialisiert werden.
- Datenstrukturen, die direkt von Hardwarebausteinen gelesen oder geschrieben werden sollen (nicht von Interfacesoftware), dürfen nicht auf dem Stack angelegt werden, sondern der Speicherplatz muß mit AllocMem() oder AllocEntry() vom System angefordert werden!!!
- Alle von mehreren Prozessen oder Tasks genutzten Datenstrukturen dürfen nicht auf dem Stack angelegt werden, sondern der Speicherplatz muß mit AllocMem() oder AllocEntry() vom System angefordert werden !!!
- Um Datenfelder von Strukturen auf NULL (0) zu initialisieren, kann bei den Funktionen AllocMem und AllocEntry das Optionflag MEMF_CLEAR gesetzt werden, um den gesamten allokierten Speicherbereich auf NULL (0) zu setzen.

Wie schon in der Einleitung gesagt, ist es für eine Multitaskingumgebung tödlich, absolute Adressen zu benutzen, da Daten und Programme gemischt mit fremden Daten und Programmen an jeder beliebigen Stelle im Speicher stehen dürfen. Die einzige absolute Adresse unter Amiga-DOS ist SYSBASE, die Speicheradresse vier, in der der Einsprungpunkt in alle Systembibliotheken eingetragen ist. Auch auf Hardwareregister sollte unbedingt nur mit Bibliotheksfunktionen zugegriffen werden. Diese extreme Flexibilität erlaubt dem Amiga und seinem Betriebssystem mit der Zeit behutsam seine Hardware und Software zu verändern und verbessern, ohne daß alte Software an das neue System angepaßt werden müßte. Programme, die unter Kickstart 1.0 unter Beachtung aller Konventionen erstellt wurden, laufen unter der aktuellen Version 1.3 immer noch ohne jede Beeinträchtigung. Alle abstürzenden Programme aus dieser Zeit beweisen nur, daß damals nur wenige Programmierer schon konsequent Betriebssystemkonform programmierten.

Niemals sollten direkt die Exception-Vektoren der CPU verändert, in den Superusermodus geschaltet oder Interrupts manipuliert werden, da diese Funktionalität massiv für die Steuerung des Multitasking genutzt wird. Ein Benutzerprozeß sollte im Usermodus des Prozessors laufen.

Alle Hardwareresourcen, wie serielle und parallele Schnittstelle oder die Soundausgabe werden von allen auf dem Amiga laufenden Prozessen geteilt. Deshalb besitzt das Betriebssystem ausgeklügelte Mechanismen, um es jedem Prozeß zu ermöglichen, auf eine bestimmte Hardware zuzugreifen ohne andere Prozesse zu stören oder von fremden Prozessen gestört zu werden.

Bevor auf von mehreren Prozessen genutzte Datenstrukturen schreibend zugegriffen werden darf, muß verhindert werden, daß andere Prozesse gleichzeitig auf diese Datenstruktur zugreifen. Über den Semaphore-Mechanismus kann ein Prozeß um Zugriff bitten, während alle anderen berechtigten Prozesse die Mitteilung bekommen, daß im Moment die Datenstruktur für die Benutzung gesperrt ist. Braucht der zugriffsberechtigte Prozeß die Datenstruktur momentan nicht mehr, gibt er sie wieder für die Benutzung durch die anderen Prozesse frei.

Falls es für ihr Programm nötig war, für kurze Zeit das Multitasking oder Interrupts abzuschalten, sollten diese Phasen so kurz wie möglich sein.

Die Statusflags des Prozessors sind nach Unterprogrammaufrufen von Betriebsystemfunktionen normalerweise nicht mehr korrekt gesetzt und führen zu falschen Ergebnissen, wenn sie ausgewertet werden.

Nur unbelehrbare Programmierer können sich nicht von alten Verhaltensweisen lösen, die auf einfachen Computern wie C64 oder IBM-PC kompatiblen Geräten unter MS-DOS genutzt werden durften und teils auch werden mußten.

2.3 Parameterübergabe beim Start eines Programms

Wenn ein Programm von der SHELL gestartet wird, wird die Eingabe in der Kommandozeile vom C-Startupmodul in einzelne Worte zerlegt. Als Wortende wird ein Leer- oder Trennungszeichen gewertet. Die für C übliche Parameterliste char *argv[] und die Anzahl der Parameter int argc werden an das Programm geliefert. Falls das Programm von der Workbench gestartet wurde, öffnet Lattice C automatisch ein CON:-Fenster für die C Standard-Ein- und Ausgabefunktionen (c.o, cres.o catch.o), bei Aztec C muß im Icon im Feld ToolTypes ein Eintrag der Form

```
"WINDOW = CON:0/11/400/100/Ich Bins"
```

vorgenommen werden, damit das Startup-Modul ein CON:-Fenster öffnet.

argv ist eine Tabelle von Zeigern, die auf die Argument-Zeichenketten zeigen.

Wird ein Programm durch das Anklicken eines Icons auf der Workbench gestartet, so erhält es keine Parameter über die Kommandozeile. argc hat in diesem Fall den Wert null (0).

Im Icon können jedoch bestimmte Variablen mit Werten belegt werden, die das Programm abfragen kann. Das kann vom Programmierer oder Benutzer geschehen, indem das Icon einmal angeklickt wird und danach der Menüpunkt Info im Workbench-Menü angewählt wird. Darauf können die Einträge im Feld ToolTypes verändert oder neue hinzugefügt werden.

Um nun ein Programm von der Workbench oder der SHELL korrekt starten zu können, sollte das Programm sowohl die Kommandozeile, als auch die ToolTypes-Einträge im zum Programm gehörigen Icon auswerten können.

```
/*************************************************************/
/* Modul:          getparams.c                               */
/*-----------------------------------------------------------*/
/* Autor:          Wilfried Häring                           */
/* Datum:          29.4.1989                                 */
/*-----------------------------------------------------------*/
/* Beschreibung:                                             */
/* Je nach dem, ob ein Programm von der Workbench oder SHELL  */
/* gestartet wurde, werden die Parameter aus der Kommandozeile */
/* oder aus dem ToolTypes-Feld des dem Progamm zugeordneten   */
/* Icon gelesen.                                             */
/*************************************************************/

 /* Benötigte Header-Dateien                                  */
#include <proto/exec.h>
#include <proto/dos.h>
#include <exec/types.h>
#include <workbench/workbench.h>
#include <workbench/startup.h>
#include <workbench/icon.h>
#include <stdio.h>
#include <string.h>

 /* Prototypen wegen fehlerhaften proto/icon.h                */
extern long atol(char *);
extern exit(int);
extern struct DiskObject *GetDiskObject(char *);
extern char * FindToolType(char **, char *);
extern long MatchToolValue(char *, char *);
extern void FreeDiskObject(struct DiskObject *);

 /* Rückgabewerte (wird nur rudimentär genutzt)               */
#define ALLRIGHT 0
#define WARN 5

 /* benötigte Version der Icon.library                        */
#define ICON_REV        0L

 /* Konstanten zum Anzeigen der gewählten Betriebsart         */
#define SCREEN          1<<0            /* entspricht -s */
#define INTERLACE       1<<1            /* entspricht -i */
#define COLORMODE       1<<2            /* entspricht -c */
```

```
#define UNITS           1<<3            /* entspricht -u */
#define DEFAULTPAGE     1<<5            /* entspricht -d */
#define USA             0
#define DINA3           1
#define DINA4           2
#define DINA5           3

 /* von OpenLibrary initialisierter Zeiger auf die            */
 /* Sprungtabelle der Icon.library                            */
struct IconBase         *IconBase;

 /* Benutzungshinweise ausgeben                               */
void usage ( void )
{
  printf ( "getparams -s -i -c<n> -u -da<n>\n" );
  Delay ( 200 );
  exit ( WARN );
}

 /* Ermöglicht es auch die Startup-Message der Workbench      */
 /* auszuwerten, wird vom Lattice Startup-Code so aufbereitet */
typedef union argv
{
  char                  **cliargv;
  struct WBStartup      *wbmsg;
} Argv;

void main ( int argc, Argv argv )
{
struct WBArg            *wb_arg;
char            **wb_argv;
int             wb_argc;
struct DiskObject       *dsk_obj;
BPTR            oldlock;
char            *tok_p;

long            actions;
long            planenum;
char            numbuf[13];
int             len;
long            pageform;

  actions = 0;
  planenum = 2;
  pageform = USA;
```

```
 /* Wenn argc == 0 wurde das Programm von der Workbench     */
 /* gestartet, sonst Kommandozeile auswerten                */
if ( argc != 0 )
{
   /* Hier werden die Zeiger auf die Kommandozeilenargumente*/
   /* direkt manipuliert. Das sollte nur gemacht werden,    */
   /* wenn man sicher ist, daß man im weiteren Verlauf des  */
   /* Programms nicht darauf zugreifen möchte!              */
  for ( argc--, argv.cliargv++;
        argc > 0;
        argc--, argv.cliargv++ )
  {
     /* Zerlegung der Argumente in einzelne Tokens, zwischen*/
     /* zwei Tokens kann ein Trennzeichen stehen, muß aber  */
     /* nicht (-iuc4)                                       */
    tok_p = *argv.cliargv;
    if ( ( *tok_p == '-' ) || ( *tok_p == '\\' ) )
    {
      for ( tok_p++; *tok_p; tok_p++ )
      {
        switch ( *tok_p )
        {
         /* fünf Beispiele herausgegriffen                  */
        case 's' :
           /* Schalter, wenn gesetzt Customscreen benutzen  */
          actions |= SCREEN;
          break;

        case 'i' :
           /* Wenn gesetzt Interlacemodus aktivieren        */
           /* geht nur auf Customscreen                     */
          actions |= INTERLACE | SCREEN;
          break;

        case 'c' :
           /* verlangt als weiteres Argument die Anzahl der */
           /* Bitplanes, die verwendet werden sollen        */
          actions |= COLORMODE;
           /* Rest dieses Arguments muß aus einer Zahl      */
           /* bestehen, Zeiger auf die Zeichen dieses       */
           /* Arguments korrekt weiterzählen                */
          len = stcd_l ( ++tok_p, &planenum );
```

```
            tok_p += ( len - 1 );
            if ( ( planenum < 1 ) || ( planenum > 24 ) )
              usage();
            continue;       /* nächstes Wort von Kommandozeile */

          case 'u' :
             /* Schalter, wenn gesetzt Maßeinheit Zentimeter  */
            actions |= UNITS;
            break;

          case 'd' :
             /* Wahlmöglichkeit zwischen amerikanischem Format*/
             /* (voreingestellt), A3 (a3) A4 (a4), A5 (a5)    */
            actions = DEFAULTPAGE;

            ++tok_p;
            if ( strcmp ( tok_p, "a3" ) == 0 )      /* gleich */
            {
              pageform = DINA3;
            }
            else if ( strcmp ( tok_p, "a4" ) == 0 )
            {
              pageform = DINA4;
            }
            else if ( strcmp ( tok_p, "a5" ) == 0 )
            {
              pageform = DINA5;
            }
            ++tok_p;
            continue;       /* nächstes Wort von Kommandozeile */

          default :
            usage();
            break;
          }
        }
      }
      else
      {
        usage();
      }
    }
  }
  /* Auswertung der Argumente von der Workbench              */
else
```

```
{
  if ( IconBase = (struct IconBase *)
                    OpenLibrary ( "icon.library", ICON_REV ) )
  {
     /* Zugriff auf die Argumente in der Workbench Startup- */
     /* Message                                             */
    for ( wb_argc = argv.wbmsg->sm_NumArgs,
          wb_arg = argv.wbmsg->sm_ArgList;
          wb_argc > 0;
          wb_argc--, wb_arg++ )
    {
       /* Programme können auch als »DefaultTool« von einem */
       /* Projekt-Icon nachgeladen werden, Sie bekommen dann*/
       /* keine weiteren Argumente                          */
      if ( ( wb_argc == 0 ) && ( wb_arg->wa_Lock == 0 ) )
      {
        printf ( "Programm nach anklicken eines Projekt-Icon
                  nachgeladen\n" );

      }
      else
      {
         /* In das korrekte Verzeichnis wechseln!          */
        oldlock = CurrentDir ( wb_arg->wa_Lock );
         /* .info-Datei von Disk lesen                     */
        if ( dsk_obj = GetDiskObject ( wb_arg->wa_Name ) )
        {
           /* ToolTypes sind wie das standard argv von der  */
           /* SHELL als Array von Zeigern auf Strings       */
           /* organisiert                                   */
          wb_argv = dsk_obj->do_ToolTypes;
           /* Wegen denkbarer Änderungen in der             */
           /* Datenstruktur von Icons gibt es               */
           /* Standardfunktionen, um auf die Daten          */
           /* zuzugreifen
          if ( tok_p = FindToolType ( wb_argv, "SCREEN" ) )
          {
            if ( MatchToolValue ( tok_p, "CUSTOM" ) )
            {
              actions |= SCREEN;
            }
          }
          if ( tok_p =
                      FindToolType ( wb_argv, "INTERLACE" ) )
          {
```

```
    if ( MatchToolValue ( tok_p, "ON" ) )
    {
      actions |= INTERLACE | SCREEN;
    }
  }
  if ( tok_p =
              FindToolType ( wb_argv, "COLORMODE" ) )
  {
     /* Krücke gebastelt                                  */
    for ( planenum = 1;
          planenum <= 24;
          planenum++ )
    {
      if ( len = stcl_d ( numbuf, planenum ) )
      {
        if ( MatchToolValue ( tok_p, numbuf ) )
        {
          actions |= COLORMODE;
          break;
        }
      }
    }
    if ( planenum > 24 )
    {
      usage();
    }
  }
  if ( tok_p =
                  FindToolType ( wb_argv, "UNITS" ) )
  {
    if ( MatchToolValue ( tok_p, "CM" ) )
    {
      actions |= UNITS;
    }
  }
  if ( tok_p =
            FindToolType ( wb_argv, "DEFAULTPAGE" ) )
  {
    actions |= DEFAULTPAGE;

    if ( MatchToolValue ( tok_p, "A3" ) )
    {
      pageform = DINA3;
    }
```

```
            else if ( MatchToolValue ( tok_p, "A4" ) )
            {
              pageform = DINA4;
            }
            else if ( MatchToolValue ( tok_p, "A5" ) )
            {
              pageform = DINA5;
            }
          }
           /* Aufräumen nicht vergessen!                    */
          FreeDiskObject ( dsk_obj );
        }
        CurrentDir ( oldlock );
      }
    }
    CloseLibrary ( (struct Library *)IconBase );
  }
  else
    printf (
           "Icon.library konnte nicht geöffnet werden!!!\n" );
}
 /* Demoausgabe der eingestellten Parameter                  */
if ( actions & SCREEN )
{
  printf ( "Programm soll Customscreen benutzen.\n" );
}
else
{
  printf ( "Programm soll Workbenchscreen benutzen.\n" );
}

if ( actions & INTERLACE )
{
  printf ( "Screen soll im Interlacemodus geöffnet werden.\n" );
}
else
{
  printf ( "Screen soll im Normalmodus geöffnet werden.\n" );
}

printf ( "Screen soll %ld Bitplanes benutzen.\n", planenum );

if ( actions & UNITS )
{
  printf ( "Als Maßeinheit werden Zentimeter benutzt.\n" );
```

```
    }
    else
    {
      printf ( "Als Maßeinheit werden Zoll benutzt.\n" );
    }

    printf ( "Als Seitengröße wurde %ld eingestellt.\n", pageform );

    Delay ( 200 );
    exit ( ALLRIGHT );
  }
```

3 Die Amiga-DOS-Datenstrukturen

Dieses Kapitel beschreibt die Datenstrukturen, die Amiga-DOS verwendet.

3.0 Einführung

Amiga-DOS stellt eine geräteunabhängige Ein- und Ausgabe zur Verfügung. Dazu wird ein Handler-Prozeß für jedes Gerät installiert, der alle Ein-/Ausgabe-Anforderungen von Amiga-DOS erkennt und diese in ein jeweils gerätetreiberspezifisches Kommando umsetzt.

Alle Ein- und Ausgaben gehen deshalb nicht direkt zum Gerätetreiber, sondern zum Handler-Prozeß. Jedoch sind auch direkte Zugriffe auf den Gerätetreiber möglich, wenn dies ausnahmsweise nötig sein sollte. Dieses Kapitel beschreibt die Datenstrukturen von Amiga-DOS, einschließlich der Prozeß-Kontrollstruktur und anderer wichtiger Datenstrukturen, sowie den Zugriffen auf einen Handler.

Amiga-DOS benutzt oft den Datentyp BPTR. Ein BPTR ist ein BCPL-Zeiger, auf die Adresse eines Langwortes geteilt durch vier. Um in C einen BPTR zu verwenden, verschieben Sie ihn einfach zweimal nach links und erhalten dadurch einen C-Zeiger (oder man verwendet das Macro BADDR() aus der Include-Datei INCLUDE:libraries/dos.h).

Um einen BPTR zu erzeugen, wird mit AllocMem() ein Speicherbereich reserviert oder eine Struktur innerhalb des Stapelspeichers angesprochen. Dazu müssen Sie aber sicher sein, nur Langworte (32 Bit) auf den Stack geschrieben zu haben (der normale Stack ist auf Worte (16 Bit) ausgerichtet), AllocMem() liefert immer Speicherbereiche zurück, die auf Langwortadressen beginnen. Der C-Zeiger wird dann zweimal nach rechts verschoben, um einen BPTR zu erzeugen.

Zeichenketten werden anders als in C nicht mit einem Nullbyte beendet, sondern das erste Byte enthält die Länge der Zeichenkette, darauf folgen die Bytes des Textes ohne ein Nullbyte als Endmarkierung. Ein BSTR ist ein BPTR auf das erste Byte (die Längenangabe) der BCPL-Zeichenkette.

In diesem Kapitel taucht ein »Global-Vector« auf. Der Global-Vector ist eine von BCPL verwendete Sprungtabelle und zeigt auf eine systemweit genutzte Standardsprungtabelle. Einige Prozesse (verschiedene in BCPL geschriebene Handler) verwenden allerdings einen eigenen Global-Vector, der vom Standard-Vector verschieden ist.

Die Definitionen für die folgenden Strukturen sind in den Dateien »INCLUDE: libraries/dos.h« und »INCLUDE:libraries/dosextens.h« für die Sprache C enthalten. Die entsprechenden Assemblerdateien heißen »dos.i« und »dosextens.i«.

3.1 Datenstrukturen für die Prozeßverwaltung

Die folgenden Datenstrukturen werden beim Starten eines Amiga-DOS-Prozesses für die Prozeßverwaltung erzeugt, für jeden Prozeß ist eine Prozeß-Kontrollstruktur vorhanden.

```
struct Process {
     /* Task-Kontrollstruktur von EXEC                        */
    struct  Task    pr_Task;
     /* MsgPort, CreateProc() liefert C-Zeiger hierauf        */
    struct  MsgPort pr_MsgPort;
     /* Alle weiteren Variablen beginnen auf Langwortaddressen*/
    WORD    pr_Pad;
     /* Array aller Segmente, die dieser Prozeß benutzt       */
    BPTR    pr_SegList;
     /* Größe des Stapelspeichers in Bytes                    */
    LONG    pr_StackSize;
     /* Global-Vector dieses Prozeß                           */
    APTR '  pr_GlobVec;
     /* Prozeßnummer eines von der SHELL gestarteten Prozeß   */
     /* null, wenn der Prozeß nicht von einer SHELL gestartet */
     /* wurde.                                                */
    LONG    pr_TaskNum;
     /* Zeiger auf das Ende des Stapelspeichers               */
    BPTR    pr_StackBase;
     /* Zweiter Rückgabewert des letzten Funktionsaufrufs,    */
     /* kann mit IoErr() abgefragt werden.                    */
    LONG    pr_Result2;
     /* Lock auf das aktuelle Verzeichnis des Prozesses          */
    BPTR    pr_CurrentDir;
     /* Standard Eingabe von der SHELL                        */
    BPTR    pr_CIS;
     /* Standard Ausgabe von der SHELL                        */
    BPTR    pr_COS;
     /* Console-Handler der Ein-/Ausgabe SHELL                */
    APTR    pr_ConsoleTask;
     /* Datei-Handler Prozeß des aktuellen Laufwerks          */
    APTR    pr_FileSystemTask;
     /* Zeiger auf die Kontrollstruktur des                   */
     /* Kommandozeileninterpreter                             */
```

```
        BPTR    pr_CLI;
         /* Zeiger auf die vorherige Stapelspeichergröße          */
        APTR    pr_ReturnAddr;
         /* Zeiger auf Funktion, die aufgerufen wird, wenn eine   */
         /* Nachricht vom Betriebssystem für den Prozeß ankommt.  */
        APTR    pr_PktWait;
         /* Standard Ausgabe für Fehlermeldungen                  */
        APTR    pr_WindowPtr;
    };  /* Process */
```

Mit pr_SegList finden Sie heraus, welche Segmente ein bestimmter Prozeß belegt. Als SegArray dient ein Feld aus Langworten, dessen eigene Länge in SegArray[0] abgelegt ist. Seine Elemente sind null oder ein BPTR zu einer SegList. CreateProc() initialisiert die Segmentliste. Dabei zeigen die ersten beiden Elemente auf residenten Code, und ist der BPTR, der von CreateProc() als Funktionswert zurückgeliefert wird. Wird ein Prozeß beendet, macht FreeMem() den Speicher für das SegArray wieder frei.

In pr_StackSize wird die Größe des Prozeß-Stapelspeichers eingetragen, der für den Prozeß mit CreateProc() bereitgestellt wird. Beachten Sie bitte, daß der Prozeß-Stapelspeicher nicht mit dem Stapelspeicher der SHELL für Programmaufrufe identisch ist. Die SHELL holt den Befehlsstapelspeicher direkt vor Start des Programmes. Seine Größe kann mit dem DOS-Befehl STACK verändert werden.

Wenn ein Prozeß explizit einen neuen Prozeß startet, erhält Amiga-DOS den Prozeß-Stapelspeicher und schreibt dessen Größe nach StackSize. Die Zeiger auf den Speicherplatz für die Prozeß-Kontrollstruktur und auf den Stapelspeicher werden ebenfalls im MemEntry-Feld der Task-Kontrollstruktur gespeichert.

Wenn ein Prozeß terminiert, wird der Speicherplatz durch Aufruf von FreeMem() wieder verfügbar gemacht. Sie können auch den Speicherplatz in die Speicherliste in der Task-Kontrollstruktur eintragen, den der Prozeß bei seinem Ablauf mit AllocMem() anfordert, damit er automatisch wieder freigegeben werden kann, wenn der Prozeß (Task) seine Arbeit beendet. Damit kann einiges an Verwaltungsaufwand im Prozeß gespart werden.

Wird ein Prozeß mit dem Aufruf von CreateProc() erstellt, zeigt der Zeiger pr_GlobVec auf den Standard-Global-Vector des Amiga. Einige interne Prozesse benutzen jedoch besondere private GlobVecs.

pr_taskNum gibt die Prozeßnummer eines Prozesses an, der von der SHELL gestartet wurde, und ist null, wenn der Prozeß nicht von einer SHELL gestartet wurde.

Der Zeiger pr_StackBase zeigt auf das physikalisch obere Ende des Prozeßstapelspeichers im RAM. Dies ist das Ende des Stack, wenn in C oder Assembler programmiert wird, in Sprachen wie BCPL ist es der Anfang des Stack.

Der Wert von pr_Result2 wird von DOS-Funktionen gesetzt, falls ein Fehler eintrat, und kann mit IoErr() abgefragt werden (bitte nur mit IoErr()!!!) und CurrentDir wird von der gleichnamigen Amiga-DOS-Funktion gesetzt.

pr_CIS und pr_COS sind File-Handles, die die Funktionen Input und Output zurückliefern. Sie sollten diese Werte bei Programmen verwenden, die unter der SHELL laufen, um die Standard-Ein- und Ausgabe festzustellen. Sonst sind pr_CIS und pr_COS Null.

pr_ConsoleTask und pr_FileSystemTask zeigen auf die Kontrollstruktur des Console-Handlers des aktuellen Fensters und den File-Handler des aktuellen Laufwerkes. Diese Werte werden vom DOS verwendet, wenn das aktuelle Console-Fenster (*) oder eine Datei durch Angabe ihres Namens relativ zum aktuellen Verzeichnis geöffnet werden soll.

Der Zeiger pr_CLI ist nur dann Null, wenn der Prozeß kein SHELL-Prozeß ist. Sonst zeigt dieser Zeiger auf eine Kommandozeileninterpreter-Kontrollstruktur:

```
struct CommandLineInterface {
     /* Zweiter Funktionswert des letzten Befehls              */
    LONG   cli_Result2;
     /* Name des aktuellen Verzeichnis                         */
    BSTR   cli_SetName;
     /* Lock auf das Verzeichnis mit den DOS-Befehlen          */
    BPTR   cli_CommandDir;
     /* Rückgabewert des letzten DOS-Befehls an die SHELL      */
    LONG   cli_ReturnCode;
     /* Name des zuletzt aufgerufenen DOS-Befehls              */
    BSTR   cli_CommandName;
     /* Abbruchschranke, wird mit DOS-Befehl FAILTAT gesetzt   */
    LONG   cli_FailLevel;
     /* Prompt (Eingabeaufforderung), die mit dem DOS-Befehl   */
     /* PROMPT eingestellt wurde                               */
    BSTR   cli_Prompt;
     /* standard Eingabe von der SHELL                         */
    BPTR   cli_StandardInput;
     /* momentane (möglicherweise durch Eingabeumleitung       */
     /* umgestellte) Eingabedatei                              */
    BPTR   cli_CurrentInput;
     /* Name der Script-Datei, wenn eine Script-Datei zur      */
     /* Ausführung gestartet wurde                             */
    BSTR   cli_CommandFile;
     /* DOSTRUE, wenn Ausgabe ein Console-Fenster ist, und     */
     /* ein Prompt ausgegeben werden muß                       */
    LONG   cli_Interactive;
     /* DOSTRUE, wenn mit RUM als Hintergrundprozeß gestartet */
    LONG   cli_Background;
     /* momentan eingestellte Ausgabedatei                     */
    BPTR   cli_CurrentOutput;
     /* Stapelspeichergröße in Anzahl der Langworte, die beim */
     /* Start eines Prozesses von der SHELL dem Prozeß         */
     /* mitgegeben werden soll                                 */
```

```
        LONG   cli_DefaultStack;
         /* standard Ausgabe auf die SHELL                          */
        BPTR   cli_StandardOutput;
         /* BPTR auf die Segmentliste des gerade ausgeführten       */
         /* SHELL-Prozesses                                         */
        BPTR   cli_Module;
    };  /* CommandLineInterface */
```

Die Funktion Exit() verwendet den Wert von pr_ReturnAddr, der auf die Rücksprungadresse im momentan aktiven User-Stapelspeicher zeigt. Endet ein Programm durch RTS bei einem leeren Stack, springt der Rechner zu der Adresse, die CreateProc() oder die SHELL auf den Stack gelegt haben. Endet ein Programm mit dem Aufruf von Exit(), verwendet Amiga-DOS diesen Zeiger, um dieselbe Rücksprungadresse zu finden.

Der Wert von pr_PktWait ist normalerweise Null. Ist er nicht gleich Null, ruft Amiga-DOS die Funktion auf, auf die pr_PktWait zeigt, wenn ein Prozeß angehalten wird, bis ein Signal den Eingang von einer Nachricht im MsgPort anzeigt. Wie die Funktion GetMsg() holt pr_PktWait eine eventuell zur Verfügung stehende Information. Mit dieser Funktion kann der Benutzer über den Standard-Process-MsgPort ankommende Mitteilungen ausfiltern, die nicht für Amiga-DOS bestimmt sind und die er selber auswerten will.

Der Wert von pr_WindowPtr wird verwendet, wenn Amiga-DOS einen Fehler findet, der normalerweise eine Aktion des Anwenders erfordert. Ein Beispiel für einen solchen Fehler ist ein Schreibversuch auf eine schreibgeschützte oder volle Diskette. Ist der Wert von WindowPtr auf –1 gesetzt, wird der Fehler als Fehlercode von einem Open- oder Write-Aufruf an das Programm zurückgemeldet. Ist der Wert Null, erzeugt Amiga-DOS einen Requester auf dem Bildschirm der Workbench, der den Anwender über den Fehler informiert und ihn auffordert, die Operation zu wiederholen oder abzubrechen. Wählt der Anwender Cancel, übergibt Amiga-DOS den Fehlercode an das Programm. Versucht er die Aktion ein zweites Mal oder legt er eine Diskette ein, beginnt Amiga-DOS ein weiteres Mal mit der Ausführung.

Setzen Sie einen positiven Wert für WindowPtr ein, verwendet Amiga-DOS diesen Wert als Zeiger auf eine Window-Struktur. Normalerweise setzt man dafür die Struktur des gerade genutzten Fensters ein. In diesem Fall schreibt Amiga-DOS die Fehlermeldung in das von Ihnen angegebene Fenster oder verwendet den Bildschirm der Workbench. Sie können den Wert des WindowPtr auch immer auf Null lassen. Verwenden Sie dann aber einen anderen Bildschirm als den der Workbench, wird eine Fehlermeldung von Ihrem selbsterstellten Bildschirm überdeckt.

Der Ausgangswert von WindowPtr wird von dem Prozeß übernommen, der das Feld erzeugt hat. Ändern Sie den Wert von WindowPtr für ein Programm, das unter der SHELL läuft, müssen Sie den vorher aktuellen Wert zwischenspeichern und nach Beendigung Ihres Programmes wieder eintragen. Andernfalls enthält der CLI-Prozeß einen WindowPtr auf ein Fenster, das nicht mehr existiert, da Ihre Programme die Resourcen, die sie während des Ablaufens vom System anfordern, beim Terminieren wieder freigeben sollten! (Siehe Anhang: Listings für ein Demoprogramm zum Starten eines Prozesses.)

3.2 Globale Datenstrukturen

Diese allgemeine Datenstruktur existiert nur ein einziges Mal, wird aber von allen Amiga-DOS-Prozessen gemeinsam benutzt. (Also nicht unbedacht darin herumschmieren; wenn es unbedingt erforderlich ist, müssen Sie ihre Programmteile mit Forbid() und Permit() einschließen. Diese Funktionen unterbinden zwischen beiden zusammengehörigen Aufrufen das Multitasking. Verwenden Sie beide Funktionen sparsam und halten Sie die Programmteile zwischen beiden Aufrufen kurz, um das Multitasking nicht für längere Zeit zu unterbinden.)

Wenn die Funktion OpenLibrary aufgerufen wird, um die DOS-Funktionsbibliothek zum Gebrauch zu öffnen, wird die Basisadresse dieser globalen Datenstruktur zurückgeliefert. Die Felder der Datenstruktur liegen in aufsteigender Reihenfolge nach dem Basiszeiger im Hauptspeicher, die Tabelle mit den Einsprungadressen in die DOS-Bibliothek liegt im Bereich der negativen Offsets vor der Basisadresse dieser Struktur.

```
struct DosLibrary {
     struct  Library dl_lib;
      /* Zeiger auf den unten beschriebenen Wurzelknoten       */
     APTR    dl_Root;
      /* Zeiger auf den Global-Vector von BCPL                 */
     APTR    dl_GV;
      /* PRIVATE Kopie des Prozessorregisterinhalts von DOS    */
     LONG    dl_A2;
     LONG    dl_A5;
     LONG    dl_A6;
 };  /* DosLibrary */

struct RootNode {
      /* Array, in dem alle zur Zeit laufenden SHELL-Prozesse  */
      /* eingetragen sind                                      */
     BPTR    rn_TaskArray;
      /* Segmentliste des SHELL-Prozeß                         */
     BPTR    rn_ConsoleSegment;
      /* Aktuelle Zeit                                         */
     struct  DateStamp rn_Time;
      /* Segmentliste für den Prozeß, der eingelegte Dsiketten */
      /* auf Fehler untersucht und (hoffentlich) als gültige   */
      /* DOS-Disketten anerkennt                               */
     BPTR    rn_RestartSeg;
      /* weist auf die DOS-Info-Struktur (Abschnitt 3.2.1 unten)*/
     BPTR    rn_Info;
      /* Segmentliste des Datei-Handlers                       */
     BPTR    rn_FileHandlerSegment;
 };  /* RootNode */
```

Der rn_TaskTable ist ein Array, dessen Größe in TaskTable[0] gespeichert wird. Der zu dem Prozeß gehörige MsgPort jeder einzelnen SHELL wird in diesem Feld gespeichert. Der Zeiger auf den MsgPort für den n-ten SHELL-Prozeß ist in TaskTable[n] gespeichert. Ein nicht belegtes Feld wird mit Null beschrieben. Die Befehle NEWCLI und RUN suchen in der TaskTable nach dem nächsten Feld mit Inhalt NULL und nehmen dessen TaskNum als Prozeßnummer für einen neu gestarteten SHELL-Prozeß.

Die rn_ConsoleSegment ist die SegList für den Code des Kommandozeileninterpreter. Run und NEWCLI verwenden diesen Wert beim Start einer neuen SHELL.

Im Feld rn_Time wird die aktuelle Zeit gespeichert. Die Werte ds_Days, ds_Minute und ds_Tick ergeben Datum und Uhrzeit. Der Wert für ds_Days ist die Anzahl an vergangenen Tagen seit dem 01. Januar 1978. Der Wert ds_Minute die Anzahl der vergangenen Minuten seit Mitternacht, ein ds_Tick entspricht dem fünfzigsten Teil einer Sekunde. Die Zeit wird jedoch nur jede Sekunde aktualisiert.

```
struct DateStamp {
    LONG        ds_Days;
    LONG        ds_Minute;
    LONG        ds_Tick;
 }; /* DateStamp */

 /* Anzahl der Ticks in einer Sekunde                    */
 #define TICKS_PER_SECOND      50
```

rn_RestartSeg weist auf die SegList für den Diskettenaktualisierungsprozeß, der immer dann aktiviert wird, wenn eine neue Diskette in ein Laufwerk einlegt wird.

3.2.1 Die Info-Substruktur

```
struct DosInfo {
      /* Name dieses Amiga in einem Netzwerk, meist null (0)     */
      /* weil kein Netzwerk installiert                          */
     BPTR    di_McName;
      /* Liste aller angeschlossenen Geräte                      */
     BPTR    di_DevInfo;
      /* Zur Zeit null (0)                                       */
     BPTR    di_Devices;
      /* Zur Zeit null (0)                                       */
     BPTR    di_Handlers;
      /* Zeiger auf den MsgPort des Netzwerkverwaltungsprozesses */
      /* ohne installiertes Netzwerk NULL                        */
     APTR    di_NetHand;
 };  /* DosInfo */
```

Die meisten Felder innerhalb dieser Substruktur sind zur Zeit noch leer, aber Commodore plant die Erweiterung des Systems in Richtung auf ein lokales Netzwerk.

di_DevInfo ist ein BPTR auf eine verkettete Liste, in der alle dem Amiga-DOS bekannten Geräte und Disketten eingetragen sind. Die Einträge haben zwei verschiedene Formate, eines für Disketten und eines für alle anderen Geräte. Jedem Gerät oder Verzeichnis, das mit ASSIGN als logisches Verzeichnis angemeldet wurde entspricht diese Form des Eintrages:

```
struct DosList {
      /* weist auf die nächste Struktur in der Liste          */
     BPTR              dol_Next;
      /* Art des Geräts siehe Konstanten unten                */
     LONG              dol_Type;
      /* Zeiger auf den MsgPort des Handler-Prozesses         */
     struct MsgPort    *dol_Task;
      /* Dateisystem-Lock oder null                           */
     BPTR              dol_Lock;
     union
     {
      /* entweder Gerät:                                      */
        struct
        {
         /* Name des Handlers, der geladen werden muß, wenn   */
         /* dol_SegList null ist (unten)                      */
        BSTR    dol_Handler;
         /* Stapelspeichergröße für den Handler-Prozeß        */
        LONG    dol_StackSize;
         /* Priorität des Handler-Prozeß                      */
        LONG    dol_Priority;
         /* FileSysStartupMsg für Disketten                   */
        ULONG   dol_Startup;
         /* Segmentliste des geladenen Handler-Prozesses      */
        BPTR    dol_SegList;
         /* BCPL Global-Vector, -1 für c/Assembler-Programme  */
        BPTR    dol_GlobVec;

        } dol_handler;

     /* oder eingelegte Diskette/Festplatte:                   */
        struct
        {
         /* Datum beim Formatieren der Diskette/Festplatte    */
        struct DateStamp        dol_VolumeDate;
```

```
        /* Locks, die nicht geschlossen wurden                    */
        BPTR                    dol_LockList;
        /* Diskettentyp: DOS, KICKSTART, NODOS,...              */
        LONG                    dol_DiskType;

        } dol_volume;

    } dol_misc;

    /* BSTR name der Diskette oder des Geräts                     */
    BSTR                dol_Name;
    };


/* Definition der Konstanten für dol_Type                         */
 #define DLT_DEVICE     0  /* Gerät
 #define DLT_DIRECTORY  1  /* assigntes Verzeichnis        */
 #define DLT_VOLUME     2  /* Name der eingelegten Disk    */
```

Das Feld dol_Next verkettet die Einträge. Bezieht sich der Eintrag auf ein Verzeichnis, zeigt dol_Task auf den Dateisystem-Handler der Diskette, auf dem sich das Verzeichnis befindet, das dol_Lock-Feld enthält einen Zeiger auf den Lock dieses Verzeichnisses.

Bezieht sich der Eintrag auf ein Gerät, kann es sich um ein residentes oder ein nicht residentes handeln. Ist es resident, identifiziert dol_Task den Handler-Prozeß, der Lock ist normalerweise Null. Ist das Device nicht resident, ist dol_Task Null und Amiga-DOS verwendet nur den Rest der aufgezeigten Struktur.

Enthält die SegList Null, ist der Code des Gerätetreibers für dieses Gerät nicht im Speicher. Das Feld dol_Handler enthält eine Zeichenkette mit dem Namen der Datei, die den Code enthält (zum Beispiel SYS:L/RAM-HANDLER). Ein Aufruf von LoadSeg() liest den Code in den Speicher und schreibt das Ergebnis in das Feld SegList.

Amiga-DOS erzeugt dann einen neuen Handler-Prozeß mit den Werten von dol_SegList, dol_StackSize und dol_Pri. Der neue Prozeß ist ein BCPL-Prozeß und benötigt einen globalen Vektor, das ist der Wert, den Sie mit GlobVec in der Mountlist angegeben haben, oder einen neuen, eigenen globalen Vektor, falls GlobVec Null ist.

Der neue Prozeß erhält eine Nachricht, die den ursprünglich vergebenen Namen, den in dol_Startup gespeicherten Wert und den Beginn des Eintrags in der Liste enthält. Der neue Handler-Prozesses schreibt dann die Adresse des MsgPort der Prozeß-Kontrollstruktur des Handler-Prozesses in das Feld dol_Task, sofern dies nötig ist.

Ist dieser Eintrag gemacht, benutzen alle folgenden Aufrufe des Geräts denselben Handler-Prozesses. Dieser Ablauf trifft zum Beispiel auf das Device RAM: zu. Wird in dol_Task kein Eintrag vorgenommen, führen weitere Aufrufe dieses Geräts immer zum Start eines neuen Handler-Prozesses. Das passiert beim Öffnen eines CON:-Fensters.

Ist der Eintrag im Typen-Feld gleich DLT_VOLUME, wird die alternative Struktur der union dol_misc benutzt.

In diesem Fall steht der Name der eingelegten Diskette im Feld dol_Name. Das Feld dol_Task verweist auf den Handler-Prozeß, wenn die Diskette (Platte immer) eingelegt ist, oder hat den Wert Null. Um gleichnamige Disketten zu unterscheiden, trägt Amiga-DOS den Zeitpunkt der Formatierung in die dol_VolumeDate ein. Bei Bedarf liest Amiga-DOS bei gleichnamigen Disketten das Erzeugungsdatum zum Unterscheiden der Disketten.

Ist eine Diskette gerade nicht in ein Laufwerk eingelegt, werden die aktuellen Locks im Feld dol_LockList gespeichert. Amiga-DOS verwendet das Feld dol_DiskType zur Identifizierung des Diskettentyps. Normalerweise ist dies eine Amiga-DOS-Diskette. Der Diskettentyp besteht aus maximal vier Zeichen, die höherwertigen Bits des Langwortes werden mit Null aufgefüllt.

3.3 Speicherverwaltung

Amiga-DOS verwaltet den gesamten Arbeitsspeicher mit Hilfe der von EXEC bereitgehaltenen Funktion AllocMem(). AllocMem() wird zum Anfordern des Speicherplatzes für alle DOS-Kontrollstrukturen verwendet. Nicht mehr benötigter Speicher wird mit Aufruf der Funktion FreeMem() wieder zur Verfügung gestellt. Jedes von Amiga-DOS angeforderte Speichersegment enthält im ersten Langwort die Länge des gesamten Datenblocks in Bytes. Der Speicherblock wird durch einen BPTR auf das zweite Langwort, das die ersten Datenbytes enthält, identifiziert. Die Struktur eines angeforderten Speicherbereichs sieht also immer so aus:

```
{
   LONG BlockSize; /* Größe des Speicherbereichs          */
   LONG FirstData; /* die ersten Datenbytes               */
   ...
}
```

3.4 Segment-Listen

Zum Erzeugen einer Segmentliste wird LoadSeg() aufgerufen. Als Ergebnis wird ein BPTR auf einen neu reservierten Speicherblock zurückgeliefert, in den Daten geladen wurden. Die Länge jedes der in der Liste aufgeführten Blöcke ist in dem Langwort eingetragen, das direkt vor dem Langwort im Speicher liegt, auf das der BPTR zeigt, den LoadSeg() zurückliefert (BADDR(bptr) – 4). Im Langwort, auf das der BPTR weist, wird ein BPTR auf das nächste Codesegment gespeichert, danach folgt im Speicherblock der geladene Code, so daß die Gesamtlänge des Blocks dann die Größe des Programmcodes plus 8 ist (vier für das Link-Feld und vier für das Size-Feld selbst).

Die SegList ist eine Liste, die durch BPTRs zusammengebunden wird. Der letzte Block enthält statt des Zeigers auf den nächsten eine Null.

```
{
   LONG BlockSize; /* Größe des Speicherbereichs            */
   LONG NextSeg;   /* BPTR zum nächsten Segment oder Null   */
   LONG FirstCode; /* Erster Wert des Binärfiles            */
}
```

3.5 Datei-Handles

Datei-Handles werden von der Amiga-DOS-Funktion Open() erzeugt. Sie werden unter anderem als Argument für die Funktionen Read() und Write() benötigt. Ein von Amiga-DOS zurückgelieferter Datei-Handle ist ein BPTR auf folgende Struktur zurück:

```
struct FileHandle {
     /* EXEC Message (Nachrichtenstruktur)                    */
    struct Message *fh_Link;
     /* MsgPort für ReplyMsg() nach erhalten eines Datenpakets */
    struct MsgPort *fh_Port;
     /* MsgPort für den Empfang von Daten, die mit PutMsg()   */
     /* an den Handler-Prozeß gesendet werden                 */
    struct MsgPort *fh_Type;
    LONG fh_Buf;
    LONG fh_Pos;
    LONG fh_End;
    LONG fh_Funcs;
 #define fh_Func1 fh_Funcs
    LONG fh_Func2;
    LONG fh_Func3;
    LONG fh_Args;
 #define fh_Arg1 fh_Args
    LONG fh_Arg2;
 }; /* FileHandle */
```

Die meisten dieser Felder werden von Amiga-DOS intern benutzt und sollten deshalb nicht modifiziert werden; lediglich das erste Feld fh_Link kann zum Zusammenbinden von Datei-Handles zu einer Liste dienen. Normale Operationen verwenden den Zeiger auf ein solches Datei-Handle nur als Parameter und kümmern sich nicht um die interne Struktur des Handles.

3.6 Locks

Das Amiga-DOS-Dateisystem nutzt eine spezielle Datenstruktur, den Lock, sehr intensiv. Diese Struktur erledigt zwei Aufgaben. Eine Funktion ist die Koordination von Lesezugriffen oder eines Schreibzugriffs auf eine Datei in der Mehrprozeßumgebung, die verhindert, daß der Inhalt von Dateien unkorrekt wird, weil mehrere Prozesse zur gleichen Zeit versuchen, Daten in einer Datei konkurrierend zu verändern. Ein »shared lock«, bei dem sich mehrere Prozesse lesend ein Verzeichnis teilen, verhindert jedoch nicht die Aktualisierung des Verzeichnisses.

Zum zweiten stellt ein Lock eine eindeutige Identifizierung für eine Datei dar. Obwohl auf eine Datei auf verschiedene Weise zugegriffen werden kann, ist ein Lock immer ein elementarer Zugriffsschlüssel zu dieser Datei. Der Lock enthält den Diskettenblock, in dem sich das Verzeichnis oder der Dateikopf befinden.

```
struct FileLock {
     /* BPTER auf den nächsten Lock                          */
    BPTR              fl_Link;
     /* Blocknummer auf der Diskette/Festplatte              */
    LONG              fl_Key;
     /* Zugriffsmodus SHARED_LOCK oder EXCLUSIVE_LOCK        */
    LONG              fl_Access;
     /* MsgPort des Handler-Prozesses                        */
    struct MsgPort *  fl_Task;
     /* BPTR auf eine DeviceList-Struktur                    */
    BPTR              fl_Volume;
};

/* Datei kann von Fremdprozessen gelesen werden              */
#define SHARED_LOCK        -2
 /* Synonym zu SHARED_LOCK                                   */
#define ACCESS_READ        -2
 /* Kein anderer Prozeß darf auf den Lock zugreifen, weil ein  */
 /* Prozeß Daten auf die Datei schreiben will                */
#define EXCLUSIVE_LOCK     -1
 /* Synonym zu EXCLUSIVE_LOCK                                */
#define ACCESS_WRITE       -1
```

Da Amiga-DOS das Feld fl_Link verwendet, um Locks in eine verkettete Liste einzufügen, sollten Sie es nicht verändern! Das Dateisystem verwendet das Feld fl_Key, um den Block auf der Diskette einzutragen, in dem das Verzeichnis oder der Dateikopf stehen. Das Feld fl_Access enthält die Information, ob es sich bei dem Lock um einen »shared-read-lock« (–2) oder einen »exclusive-write-lock« (–1) handelt. Das Feld fl_task enthält einen Zeiger zu dem MsgPort des Handler-Prozesses des Geräts, auf dem die Datei zu finden ist. Das Feld fl_Volume verweist auf den Eintrag in der DevInfo-Struktur, der zur eingelegten

Diskette gehört, auf die sich der Lock bezieht. Einträge zu einer Diskette in der DevInfo-Struktur enthalten Informationen, ob die Diskette eingelegt ist oder Locks auf sie offen sind.

Beachten Sie bitte, daß ein Lock auch Null sein kann. In diesem besonderen Fall zeigt ein Lock auf das Wurzelverzeichnis der Diskette oder Festplatte von der gebootet wurde. Das Feld fl_Task zeigt auf den MsgPort des Handler-Prozesses, der das Laufwerk steuert, in dem die Diskette eingelegt ist, oder das Festplattenlaufwerk.

```
/* Examine() und ExInfo() füllen die Struktur FileInfoBlock   */
/* mit Daten                                                  */

struct FileInfoBlock {
   LONG  fib_DiskKey;
    /* Wenn größer null Verzeichnis, sonst normale Datei      */
   LONG  fib_DirEntryType;
    /* Max 30 Buchstaben langer Dateiname                     */
   char  fib_FileName[108];
    /* Schutzflags rwedspa                                    */
   LONG  fib_Protection;
   LONG  fib_EntryType;
    /* Anzahl der Bytes in der Datei                          */
   LONG  fib_Size;
    /* Anzahl der Blocks in der Datei                         */
   LONG  fib_NumBlocks;
    /* Datum der letzten Änderung                             */
   struct DateStamp fib_Date;
    /* Dateikommentar (max 80 Zeichen)                        */
   char  fib_Comment[80];
    /* reserviert!!!                                          */
   char  fib_Reserved[36];
}; /* FileInfoBlock */

/* FIB meint FileInfoBlock */

/* Rückgabe von der Funktion Info(), muß auf Langwortaddresse */
/* liegen                                                     */

struct InfoData {
    /* Anzahl der Softerrors auf der Diskette                 */
   LONG  id_NumSoftErrors;
    /* Nummer der Diskettenstation, in der die Diskette       */
    /* eingelegt ist oder war.                                */
   LONG  id_UnitNumber;
    /* Siehe Konstantendefinitionen unten                     */
   LONG  id_DiskState;
    /* Gesamtanzahl der Blöcke auf der Diskette               */
```

```
    LONG  id_NumBlocks;
     /* Anzahl der benutzten Blöcke                          */
    LONG  id_NumBlocksUsed;
     /* Anzahl der Bytes pro Block                           */
    LONG  id_BytesPerBlock;
     /* Siehe Konstantendefinitionen unten                   */
    LONG  id_DiskType;
     /* BPTR auf die zugehörige volume node                  */
    BPTR  id_VolumeNode;
     /* null (0) wenn nicht auf die Diskette zugegriffen wird  */
    LONG  id_InUse;
}; /* InfoData */

/* ID meint InfoData */

/*** id_DiskState ***/
 /* Diskette ist schreibgeschützt                            */
#define ID_WRITE_PROTECTED 80
 /* Diskette wird im moment validiert                        */
#define ID_VALIDATING      81
 /* Diskette ist in Ordnung (normal)                         */
#define ID_VALIDATED       82

/*** id_DiskType ***/
 /* keine Diskette eingelegt                                 */
#define ID_NO_DISK_PRESENT (-1)
 /* Diskette unlesbar                                        */
#define ID_UNREADABLE_DISK (((long)'B'<<24) | ((long)'A'<<16)
| ('D'<<8))
 /* AmigaDOS Diskette                                        */
#define ID_DOS_DISK        (((long)'D'<<24) | ((long)'O'<<16)
| ('S'<<8))
 /* Auf AmigaDOS basierendes Fremdformat, z.B. Quarterback   */
#define ID_NOT_REALLY_DOS  (((long)'N'<<24) | ((long)'D'<<16)
| ('O'<<8) | ('S'))
 /* Kickstart Diskette vom Amiga 1000                        */
#define ID_KICKSTART_DISK  (((long)'K'<<24) | ((long)'I'<<16)
| ('C'<<8) | ('K'))


/*** Fehlermeldungen von IoErr() ***/

#define ERROR_NO_FREE_STORE              103
#define ERROR_TASK_TABLE_FULL            105
#define ERROR_LINE_TOO_LONG              120
#define ERROR_FILE_NOT_OBJECT            121
#define ERROR_INVALID_RESIDENT_LIBRARY   122
```

```
#define ERROR_NO_DEFAULT_DIR            201
#define ERROR_OBJECT_IN_USE             202
#define ERROR_OBJECT_EXISTS             203
#define ERROR_DIR_NOT_FOUND             204
#define ERROR_OBJECT_NOT_FOUND          205
#define ERROR_BAD_STREAM_NAME           206
#define ERROR_OBJECT_TOO_LARGE          207
#define ERROR_ACTION_NOT_KNOWN          209
#define ERROR_INVALID_COMPONENT_NAME    210
#define ERROR_INVALID_LOCK              211
#define ERROR_OBJECT_WRONG_TYPE         212
#define ERROR_DISK_NOT_VALIDATED        213
#define ERROR_DISK_WRITE_PROTECTED      214
#define ERROR_RENAME_ACROSS_DEVICES     215
#define ERROR_DIRECTORY_NOT_EMPTY       216
#define ERROR_TOO_MANY_LEVELS           217
#define ERROR_DEVICE_NOT_MOUNTED        218
#define ERROR_SEEK_ERROR                219
#define ERROR_COMMENT_TOO_BIG           220
#define ERROR_DISK_FULL                 221
#define ERROR_DELETE_PROTECTED          222
#define ERROR_WRITE_PROTECTED           223
#define ERROR_READ_PROTECTED            224
#define ERROR_NOT_A_DOS_DISK            225
#define ERROR_NO_DISK                   226
#define ERROR_NO_MORE_ENTRIES           232

/* Rückgabewerte für DOS-Befehle                                  */
 /* Erfolg, alles in Ordnung                                       */
#define RETURN_OK      0
 /* Warnung, kleinerer Fehler                                      */
#define RETURN_WARN    5
 /* Fehler aufgetreten                                             */
#define RETURN_ERROR  10
 /* Alles ging schief, schwerer Fehler                             */
#define RETURN_FAIL   20
```

4 DOS-Bibliotheksfunktionen

Das Amiga-DOS stellt dem Programmierer eine Bibliothek von Funktionen zur Verfügung, die für die Ein- und Ausgabe, die Dateiverwaltung, die Prozeßverwaltung, das Laden und Starten von Programmen und die DOS-interne Nachrichtenübermittlung genutzt werden können. Sie sind nicht portabel, das heißt andere Betriebssysteme wie UNIX oder OS/2 besitzen oft eine ähnliche Schnittstelle, benutzen aber andere Namen und Parameterlisten für die Funktionsaufrufe.

Um den Überarbeitungsaufwand niedrig zu halten, wenn Programme vom Amiga auf andere Betriebssysteme übertragen werden sollen, ist es daher oft sinnvoller, die genormten High-Level-Funktionen zu benutzen, die in der Standardbibliothek jedes C-Compilers mitgeliefert werden.

Die DOS-Funktionen liefern oft die Konstanten DOSTRUE (–1L) für Erfolg und DOSFALSE (0L) für Mißerfolg zurück. Die Funktion IoErr() liefert dann genauere Informationen über einen aufgetretenen Fehler.

Die Register, die zur Parameterübergabe genutzt werden, sind den einzelnen Parametern vorangestellt.

Eine alphabetische Auflistung der DOS-Funktionen:

Name: **Close**

Aufruf: Close (<D1> datei);
BPTR datei;
(struct FileHandle *)BADDR (datei);

Prototyp: void Close(BPTR);

Funktion: Eine Datei (Datenstrom) wird geschlossen. Es liegt in der Verantwortung des Programmierers, alle Dateien zu schließen, die er geöffnet hat, da Amiga-DOS keine Buchführung über die von einer Applikation geöffneten Resourcen führt. Jede geöffnete Datei darf nur genau einmal geschlossen werden. Schließen Sie deshalb niemals eine Datei, die Sie nicht geöffnet haben, sondern die Ihren Funktionen beim Aufruf mitgegeben wurde, da die aufrufende Funktion die alleinige Verantwortung für diese offene Datei hat und deshalb diese Datei auch wieder schließen wird. Wird versucht, eine Datei mehrmals zu schließen, so meldet sich Amiga-DOS mit einer Guru-Meditation ab.

Eingabe(n): datei ist ein BCPL-Zeiger auf eine Struktur des Typs FileHandle.

Funktionswert: keiner (void).

siehe auch: *Open*

Name: **CreateDir**

Aufruf: <D0> lock = CreateDir (<D1> name);
BPTR lock;
(struct FileLock *)BADDR (lock);
char *name;

Prototyp: BPTR CreateDir(char *);

Funktion: CreateDir erzeugt im aktuellen Verzeichnis oder im vollständig angegebenen Pfad ein neues Unterverzeichnis mit der Bezeichnung name. Verzeichnisse können nur auf Geräten erzeugt werden, die sie unterstützen (zum Beispiel Disketten oder Festplatten). CreateDir gibt einen BPTR auf eine FileLock-Struktur für exklusiven Zugriff zurück. Das heißt, das Programm, das das Verzeichnis angelegt hat, ist allein Schreib- und Leseberechtigt für das Verzeichnis. Wird Null zurückgegeben, konnte das Verzeichnis nicht erzeugt werden. Für eine genauere Fehleranalyse sollte dann IoErr() aufgerufen werden.

Eingabe(n): name ist ein Zeiger auf einen C-String, das heißt eine Zeichenkette, deren Ende mit einem Nullbyte gekennzeichnet ist.

Funktionswert: lock ist eine BCPL-Zeiger auf eine Struktur FileLock.

Name: **CreateProc**

Aufruf: <D0> messageport = CreateProc (<D1> name, <D2> pri, <D3> segment, <D3> stackgroesse);
struct MsgPort *messageport;
char *name;
long pri;
BPTR segment;
long stackgroesse;

Prototyp: struct MsgPort *CreateProc(char*, long, BPRT, long);

Funktion: CreateProc erzeugt einen neuen Amiga-DOS-Prozeß mit dem Namen name.

Um unter Amiga-DOS ein parallel laufendes Programm zu starten, gibt es zwei Möglichkeiten. Mit AddTask kann eine beliebige Routine gestartet und die Systemliste der vorhandenen Tasks eingebunden werden. Der Scheduler von Exec teilt dann die vorhandene

Rechenkapazität unter alle Tasks auf. Von einem Task aus können alle Amiga-Bibliotheksfunktionen aufgerufen werden, außer die Funktionen von Amiga-DOS.

Amiga-DOS verlangt zur Kommunikation mit einem autonom laufenden Programm mehr Daten, als die Kontrollstruktur eines Task liefern kann. Deshalb muß eine autonome Routine, die die DOS-Funktionen aufrufen will, als Prozeß gestartet werden. Ein Prozeß unterscheidet sich von einem Task dadurch, daß die Task-Kontrollstruktur um einen »message port« und andere Daten erweitert wird, die zum Beispiel beschreiben, in welcher Weise der Prozeß mit der Console (Tastatur und Mauseingabe) verbunden ist, und wo sich das Ausgabefenster für diese Console befindet.

Um einen Prozeß zu starten muß vorher der Programmcode mit der Funktion LoadSeg() geladen werden. LoadSeg liefert den BCPL-Zeiger segment auf die Segmentliste zurück, wohin der Programmcode geladen wurde. Der erste Programmblock (hunk) wird angesprungen und ausgeführt und muß deshalb alle notwendigen Initialisierungen vornehmen. C-Programmierer brauchen sich hierum nicht explizit zu kümmern. Eine Prozeß-Kontrollstruktur wird von CreateProc bereitgestellt und initialisiert.

Die Variable stackgroesse übergibt die Größe des Stapelspeichers, den der neue Prozeß nutzen kann, und pri übergibt die Priorität, mit der der Prozeß von Exec gestartet werden soll. name ist der Name, den der neue Prozeß erhalten soll.

Als Ergebnis liefert die Funktion einen Zeiger auf die MsgPort-Struktur innerhalb der Prozeß-Kontrollstruktur zurück, oder NULL, wenn der Prozeß nicht gestartet werden konnte.

Eingabe(n): name ist ein Zeiger auf einen, mit einem Nullbyte abgeschlossenen C-String. pri ist eine vorzeichenbehaftete long int. segment ist ein BCPL-Zeiger auf eine Segmentliste. stackgroesse ist eine long int und muß ein Vielfaches von vier sein.

Funktionswert: messageport ist ein Zeiger auf die MsgPort-Struktur in der Prozeß-Kontrollstruktur.

Beispiel: Siehe Anhang A, Quelltexte

siehe auch: LoadSeg().

Name: **CurrentDir**

Aufruf:

```
<D0> oldlock = CurrentDir (<D1> lock)
BPTR oldlock, lock;
(struct FileLock *)BADDR (oldlock);
(struct FileLock *)BADDR (lock);
```

Prototyp:

```
BPTR CurrentDir (BPTR);
```

Funktion: CurrentDir setzt dasjenige Verzeichnis als aktuelles Verzeichnis, auf das die Variable lock zeigt. Ein Zeiger auf das verlassene aktuelle Verzeichnis wird zurückgegeben. Ein Rückgabewert von null ist bei dieser Funktion ein reguläres, zulässiges Ergebnis, da ein Nullzeiger auf das Wurzelverzeichnis des Dateisystems weist, von dem das System gebootet wurde. (Das Wurzelverzeichnis des Bootgeräts ist also implizit der Vater aller weiteren Wurzelverzeichnisse der angeschlossenen Geräte.)

Eingabe(n): lock ist ein BCPL-Zeiger auf eine Struktur FileLock.

Funktionswert: oldlock ist ein BCPL-Zeiger auf eine Struktur FileLock.

siehe auch: Lock().

Name: **DateStamp**

Aufruf: ret_vec = DateStamp (vector);
long *ret_vec;
long *vector;
long vector[3];

Prototyp: long *DateStamp(long *);

Funktion: DateStamp akzeptiert als Argument einen Zeiger auf ein Feld von long-Variablen mit drei Elementen. In diese drei Elemente des Feldes werden in aufsteigender Reihenfolge die Anzahl der Tage, die seit der Datumsbasis (1.1.1978) des Systems vergangen sind, in das zweite Element die Anzahl der Minuten, die an diesem Tag vergangen sind und in das letzte Element die Anzahl von 1/50 Sekunden, die in dieser Minute vergangen sind, eingetragen.

Eingabe(n): vector ist ein Zeiger auf ein Feld mit drei Elementen des Typs long. (Ausrichtung auf Langwortadresse!)

Funktionswert: rec_vec Gibt den Zeiger auf das, wie beschrieben, gefüllte Feld zurück.

Name: **Delay**

Aufruf: Delay (<D1> ticks);
long ticks;

Prototyp: void Delay(long);

Funktion: Bringt den Prozeß für die Anzahl von ticks 1/50 Sekunden in den Wartezustand, die weitere Abarbeitung des Prozesses wird vom Scheduler nach Ablauf der Wartezeit wieder gestartet.

Bugs: Durch einen Fehler des TimerDevice in den Versionen 1.2 und 1.3 des Betriebssystems funktionieren die Floppylaufwerke und das TimerDevice nicht mehr korrekt, wenn Delay() mit dem Argument 0 aufgerufen wird.

Eingabe(n): ticks long int.

Funktionswert: keiner (void).

Name: **DeleteFile**

Aufruf: <D0> ergebnis = DeleteFile (<D1> name);
long ergebnis;
char *name;

Prototyp: long DeleteFile(char *);

Funktion: DeleteFile() versucht eine Datei oder ein Verzeichnis angegebenen Namens zu löschen. Konnte die Datei gelöscht werden, so wird der Wert DOSTRUE zurückgeliefert, sonst DOSFALSE. Um ein Verzeichnis zu löschen, müssen alle Dateien und Unterverzeichnisse in diesem Verzeichnis gelöscht sein. Eine genaue Fehlerbeschreibung kann mit IoErr() abgefragt werden.

Eingabe(n): name ist Zeiger auf einen nullterminierten C-String.

Funktionswert: long int, logischer Ausdruck.

Beispiel:

```
        ...
if (DeleteFile ("Harribald"))
{
  /* Jubel, Jubel */
}
else
{
  /* Zicke, Zacke, Hühnerkacke!*/
  error = IoErr();
  /* Außer Fluchen noch irgendwas sinnvolles tun... */
}
        ...
```

Siehe auch: IoErr();

Name: **DeviceProc**

Aufruf: <D0> messageport = DeviceProc (<D1> name);
struct MsgPort *messageport;
char *name;

Prototyp: struct MsgPort *DeviceProc(char *);

Funktion: DeviceProc gibt einen Zeiger auf den MsgPort des Prozesses zurück, der das Gerät name steuert. Wird kein entsprechender Prozeß gefunden, weil das Gerät zum Beispiel nicht angeschlossen oder nicht gemountet ist, wird DOSFALSE zurückgegeben. Wenn name auf ein aktives, gemountetes Device verweist, liefert IoErr() einen Lock auf ein Directory auf diesem Device zurück.

Eingabe(n):name Zeiger auf einen null terminierten C-String.

Funktionswert: messageport ist ein Zeiger auf den MsgPort der Prozeß-Kontrollstruktur.

Name: **DupLock**

Aufruf: <D0> lock = DupLock (<D1> oldlock);
BPTR lock, oldlock;
(struct FileLock *)BADDR (lock);
(struct FileLock *)BADDR (oldlock);

Prototyp: BPTR DupLock(BPTR);

Funktion: Der Aufruf der Funktion DupLock() ist der korrekte und einzig erlaubte Weg, um einen FileLock mit einem anderen Prozeß zu teilen. Als Parameter erwartet die Funktion einen shared-Lock (shared filing system lock). Write-Locks können nicht dupliziert werden!

Wenn der Lock nicht kopiert werden konnte, wird DOSFALSE zurückgeliefert.

Eingabe(n): oldlock ist ein BCPL-Zeiger auf eine FileLock-Struktur.

Funktionswert: lock ist ein BCPL-Zeiger auf eine FileLock-Struktur.

Siehe auch: Eine genaue Beschreibung von einem Lock erfolgt im Abschnitt über Lock();

Name: **Examine**

Aufruf: <D0> ergebnis = Examine (<D1> lock, <D2> dateiinfoblock);
long ergebnis;
BPTR lock;
(struct FileLock *)BADDR (lock);
struct FileInfoBlock *dateiinfoblock;

Prototyp: long Examine(BPTR, struct FileInfoBlock *);

Funktion: Examine() füllt die FileInfoBlock-Struktur, auf die es einen Zeiger als Parameter übergeben bekommt, mit Daten über eine Datei oder ein Verzeichnis. In dem FileInfoBlock wird der Name, die Größe, das Erzeugungsdatum und ob der Lock auf eine Datei oder ein Verzeichnis zeigt eingetragen. Die Struktur FileInfoBlock muß wegen des BCPL Codes auf einer Langwortaddresse beginnen und sollte daher mit AllocMem() aus EXEC bereitgestellt werden.

Ein Programm darf sich eine lokale Kopie der Werte eines FileInfoBlocks behalten, solange diese Werte niemals an das Betriebssystem zurückgegeben werden. Eine solche Handlung ist ein guter Weg, den Inhalt einer Diskette oder Festplatte zu Müll nachzubearbeiten.

Eingabe(n): lock ist ein BCPL-Zeiger auf eine FileLock-Struktur. dateiinfoblock ist ein Zeiger auf eine FileInfoBlock-Struktur, für die der Programmierer Platz bereitgestellt hat und die auf einer Langwortaddresse beginnt.

Funktionswert: ergebnis long int

Name: **Execute**

Aufruf: <D0>ergebnis = Execute(<D1> kommandozeile, <D2> eingabe, <D3> ausgabe)

long ergebnis;

char *kommandozeile;

BPTR eingabe, ausgabe;

(struct FileHandle *)BADDR (eingabe);

(struct FileHandle *)BADDR (ausgabe);

Prototyp: long Execute(char *, BPTR, BPTR);

Funktion: Execute() versucht die Zeichenkette kommandozeile auszuführen, als sei sie auf der SHELL wie ein DOS-Befehl mit Argumenten eingegeben worden. Die Zeichenkette muß den für den Kommandozeileninterpreter gültigen Regeln folgen, darf also auch Ein- und Ausgabeumleitung (»<« oder »>«) beinhalten.

Normalerweise wird das Eingabedatei-Handle auf NULL gesetzt, und die Kommandozeile wird als Eingabe für den DOS-Befehl ausgewertet. Es besteht jedoch auch die Möglichkeit das Datei-Handle auf eine bestimmte Eingabedatei zu setzen, von der weitere Eingaben gelesen werden, die an die Kommandozeile angefügt werden, bis das Ende dieser Eingabedatei erreicht ist.

Für die Ausgaben der DOS-Befehle wird normalerweise ein Datei-Handle übergeben, das auf eine Ausgabedatei, zum Beispiel ein Console-Fenster, zeigt. Wird NULL angegeben, wird die Ausgabe in das aktuelle Fenster geleitet, wenn keine Ausgabeumleitung angegeben wurde. Beachten Sie dabei, daß ein von der Workbench gestartetes Programm normalerweise kein aktuelles Ausgabefenster besitzt.

Mit Execute() kann auch eine neue interaktive SHELL gestartet werden, indem Execute() mit einer leeren »kommandozeile« und einem Datei-Handle aufgerufen wird, das auf ein neues Console-Fenster zeigt, das zum Beispiel mit Open() geöffnet wurde. Die neue SHELL liest dann ihre Eingaben von dem neuen Console-Fenster als Standard-Eingabe und benutzt es auch als Standard-Ausgabe. Die neue SHELL kann nur durch ENDCLI wieder geschlossen werden.

Damit Execute() arbeitet, muß der DOS-Befehl RUN auf dem logischen Gerät C: vorhanden sein. Vorsicht: Befehle, die auch von einer normalen SHELL aus nicht korrekt mit RUN zu starten sind, können auch nicht mit Execute() benutzt werden. (Zum Beispiel der DOS-Befehl EXECUTE zum Ausführen von Script-Dateien.)

Mit dem Rückgabewert DOSFALSE zeigt Execute() einen Fehler bei der Ausführung an, mit DOSTRUE Erfolg.

Eingabe(n): kommandozeile ist ein Zeiger auf einen null terminierten C-String. eingabe ist ein BPTR auf eine Struktur File-Handle. ausgabe ist ein BPTR auf eine Struktur File-Handle.

Funktionswert: ergebnis long int.

Name: **Exit**

Aufruf: Exit

Exit (<D1> rueckgabewert);

long rueckgabewert;

Prototyp: void Exit(long);

Funktion: Exit() ist nur für Programme in der Sprache BCPL oder solche, die das Verhalten von Programmen, die in BCPL geschrieben sind, simulieren, nützlich.

Um zu terminieren sollten normale C-Programme die Standardfunktion exit() benutzen (bemerken Sie bitte den Kleinbuchstaben »e« in exit()). Assemblerprogrammierer sollten einen Rückgabewert in das Register D0 laden und einen RTS ausführen.

Exit() bewirkt je nach Startumgebung des Programms etwas verschiedenes. Wurde das Programm von einer SHELL aus gestartet, beendet Exit() das Programm und kehrt zur SHELL zurück. In diesem Fall kann der rueckgabewert zum Beispiel in Script-Dateien ausgewertet werden. Wenn das Programm als unabhängiger Prozeß gestartet wurde, zum Beispiel von der Workbench, beendet Exit() den gesamten Prozeß und gibt den Speicherplatz und Stapelspeicher des Prozesses frei.

Eingabe(n): rueckgabewert long int.

Funktionswert: keiner (void).

Name: **ExNext**

Aufruf: <D0> ergebnis = ExNext (<D1> lock, <D2> dateiinfoblock);
long ergebnis;
BPTR lock;
(struct FileLock *)BADDR (lock);
struct FileInfoBlock *dateiinfoblock;

Prototyp: long ExNext(BPTR, struct FileInfoBlock *);

Funktion: ExNext() erhält als Parameter einen Lock auf ein Verzeichnis und einen Zeiger auf eine Struktur FileInfoBlock, die vorher mit Examine() initialisiert wurde, oder von einem vorausgegangenen Aufruf von ExNext aktualisiert wurde. ExNext() gibt DOSFALSE zurück, wenn ein Fehler eintrat. Meistens wird als Fehlerbedingung das Ende der Liste der Einträge in diesem Verzeichnis erreicht. In diesem Fall liefert IoErr() die Konstante ERROR_NO_MORE_ENTRIES zurück.

Folgende Schritte sind also zu tun:

- Durch einen Aufruf von Examine() wird eine Struktur FileInfoBlock mit den Werten des Verzeichnisses initialisiert, das untersucht werden soll.
- ExNext() wird zum ersten Mal mit diesen beiden Parametern untersucht.
- Die Information, die ExNext() in der Struktur FileInfoBlock zurückliefert, kann verarbeitet werden. Das strukturangehörige Feld field enthält einen positiven Wert, wenn als nächster Eintrag ein Unterverzeichnis gefunden wurde und ist negativ, wenn der Eintrag eine Datei ist.
- Solange ExNext() aufrufen, bis es DOSFALSE zurückliefert, und mit IoErr() untersuchen, ob tatsächlich der Wert ERROR_NO_MORE_ENTRIES zurückgegeben wurde und kein weiterer Eintrag im Verzeichnis vorhanden ist.

Wenn der Verzeichnisbaum rekursiv durchwandert werden soll, muß für jeden Abstieg in der Verzeichnishierarchie das neu gefundene Unterverzeichnis mit Lock() gelockt werden und dann mit Examine ein neuer(!) FileInfoBlock initialisiert werden. Wird der alte FileInfoBlock mit Examine auf die Werte des Unterverzeichnisses neu initialisiert, so werden alle Informationen über das übergeordnete Verzeichnis verloren und müssen bei Rückkehr auf diese Ebene neu initialisiert werden.

Es ist zwar zulässig, den Lock auf das alte Verzeichnis mit Unlock() wieder aufzuheben und später mit Lock wieder zurückzuholen, kostet aber sehr viel Laufzeit, da bei der Rückkehr die verlorengegangene Information wieder vollständig restauriert werden muß.

Es ist nicht zulässig, eine Datei mit Examine zu untersuchen und von dieser Datei aus mit ExNext() den Rest des Verzeichnisses zu untersuchen!

Man kann den FileInfoBlock in eine private Struktur kopieren. Diese private Kopie darf jedoch niemals an das Betriebssystem zurückgegeben werden, da in einem Multitaskingsystem jederzeit ein anderer Prozeß die Verzeichnisstruktur ändern könnte.

Eingabe(n): lock ist ein BCPL-Zeiger auf eine Struktur FileLock vom initialisierenden Aufruf von Examine(). dateiinfoblock ist ein Zeiger auf eine Struktur FileInfoBlock, die von Examine() initialisiert wurde.

Funktionswert: ergebnis long int.

Siehe auch: Examine(), Lock().

Name: **Info**

Aufruf: <D0> ergebnis = Info (<D1> lock, <D2> parameterblock);
BPTR lock;
(struct FileLock *)BADDR (lock);
struct InfoData *parameterblock;
long ergebnis;

Prototyp: long Info(BPTR, struct InfoData *);

Funktion: Info() dient zum Abfragen von Informationen über alle eingelegten Disketten oder aktiven Festplattenpartitionen. Die Variable lock kann entweder auf das Gerät oder eine Datei auf dem Gerät verweisen. Die Variable parameterblock zeigt auf einen vorher mit AllocMem() angeforderten Speicherbereich für eine Struktur InfoData, die von Info() mit Werten gefüllt wird. In der Struktur InfoData werden Informationen über die Speicherkapazität der Diskette oder Festplatte, die Anzahl der freien Blöcke und die Fehler zurückgegeben.

Konnte keine Information von der Disk abgefragt werden, wird DOSFALSE zurückgeliefert, sonst DOSTRUE.

Eingabe(n): lock ist ein BCPL-Zeiger auf eine Struktur FileLock. parameterblock ist ein Zeiger auf eine Struktur InfoData, die auf einer Langwortaddresse beginnen muß.

Funktionswert: ergebnis long int.

Name: **IoErr**

Aufruf: <D0> fehler = IoErr();
long fehler;

Prototyp: long IoErr(void);

Funktion: Wenn DOS-Funktionen eine Fehlerbedingung finden, geben sie den Wert DOSFALSE zurück. In diesem Fall kann mit IoErr() eine genauere Fehlerbeschreibung abgeholt werden. Manche Funktionen nutzen IoErr() auch, um ein weiteres Ergebnis zurückzuliefern.

Eingabe(n): keine (void).

Funktionswert: fehler int.

Siehe auch: Open(), Read(), ExNext().

Name: **Input**

Aufruf: <D0> datei = Input();,
BPTR datei;
(struct FileHandle *)BADDR (datei);

Prototyp: BPTR Input(void);

Funktion: Input() gibt ein Datei-Handle auf den Eingabedatenstrom zurück, der beim Starten des Programms angelegt wurde.

Eingabe(n): keine (void).

Funktionswert: datei ist eine BCPL-Zeiger auf eine Struktur FileHandle.

Siehe auch: Output().

Name: **IsInteractive**

Aufruf: <D0> ergebnis = IsInteractive (<D1> datei);
long ergebnis;
BPTR datei;
(FileHandle *)BADDR (datei);

Prototyp: long IsInteractive(BPTR);

Funktion: IsInteractive() hilft festzustellen, ob eine Datei mit einem Console-Fenster verbunden ist, in dem interaktiv vom Benutzer Daten eingegeben und auf das der Prozeß Daten schreiben kann. Der Rückgabewert DOSTRUE zeigt an, daß die mit dem Datei-Handle datei verbundene Datei ein interaktives Terminal ist, DOSFALSE signalisiert, daß datei zum Beispiel mit einer Datei des Filing-Systems verbunden ist.

Eingabe(n): datei ist ein BCPL-Zeiger auf eine Struktur FileHandle.

Funktionswert: ergebnis long int.

Name: **LoadSeg**

Aufruf: <D0> segment = LoadSeg (<D1> name);
BPTR segment;
char* name;

Prototyp: BPTR LoadSegment(name*);

Funktion: Mit LoadSeg() kann eine ausführbare, von einem Linker erzeugte Datei in den Speicher des Amiga geladen werden. Die drei Segmente Code, initialisierte Daten und Speicherplatz für uninitialisierte Daten werden in eine Liste eingetragen, die mit BCPL-Zeigern verkettet ist, das Ende der Liste wird mit einem Null-Zeiger gekennzeichnet.

Falls beim Laden irgendein Fehler geschah, werden alle schon geladenen Segmente wieder aus dem Speicher entfernt und die Funktion gibt DOSFALSE (null) zurück.

Wenn das Programm korrekt in den Speicher geladen werden konnte, gibt die Funktion einen Zeiger auf den Beginn der Segmentliste zurück. Programmcode kann mit UnLoadSeg() wieder aus dem Speicher entfernt werden.

Eingabe(n): name ist ein Zeiger auf einen null terminierten C-String, den Namen des zu ladenden Programms.

Funktionswert: segment ist ein BCPL-Zeiger auf den Anfang der Segmentliste.

Siehe auch: UnLoadSeg(), CreateProc().

Name: **Lock**

Aufruf: <D0> lock = Lock (<D1> name, <D2> zugriffsweise);
BPTR lock;
(struct FileLock *)BADDR (lock);
char *name;
long zugriffsweise;

Prototyp: BPTR Lock(char*, long);

Funktion: Lock() gibt einen Lock auf ein Verzeichnis oder eine Datei name zurück, das dem Benutzer den Zugriff auf die Datei oder das Verzeichnis erlaubt, sonst null.

Es gibt zwei Zugriffsmöglichkeiten auf eine Datei:

- ACCESS_READ läßt einen gemeinsamen lesenden Zugriff von mehreren Prozessen auf eine Datei zu, da keiner der Prozesse dabei den Inhalt der Datei verändern kann.
- ACCESS_WRITE gibt einem Prozeß das alleinige Zugriffsrecht auf die Datei, damit andere Prozesse nicht aus dem Takt kommen, wenn der Inhalt der Datei verändert wird, während sie versuchen sie zu lesen oder zu schreiben.

Die Laufzeit eines Aufrufes von Lock() ist geringer, als der von Open(), so daß zum Testen, ob eine Datei existiert, Lock() die effektivere Wahl ist. Wenn sichergestellt ist, das eine Datei existiert und sie geöffnet werden soll, muß dazu Open() verwendet werden.

Eingabe: name ist ein Zeiger auf einen null terminierten C-String.
zugriffsweise long int.

Funktionswert: lock ist ein BCPL-Zeiger auf eine Struktur FileLock.

Name: **Open**

Aufruf: <D0> datei = Open (<D1> name, <D2> zugriffsweise);
BPTR datei;
(struct FileHandle *)BADDR (datei);
char *name;
long zugriffsweise;

Funktion: Open() öffnet die benannte Datei und gibt den dazugehörenden Datei-Handle zurück.

Es gibt mehrere erlaubte Zugriffsweisen auf eine Datei:

- MODE_OLDFILE öffnet eine existierende Datei zum Lesen und Schreiben und setzt den Positionszeiger in der Datei auf den Dateianfang.
- MODE_NEWFILE erzeugt eine neue Datei zum Lesen und Schreiben, eine alte Datei gleichen Namens wird gelöscht.
- MODE_READWRITE öffnet eine existierende Datei zum Schreiben mit einem exclusiven Lock.

Der name kann auf einen Dateinamen (optional mit vorangestellten Gerätenamen), auf ein einfaches Gerät wie NIL:, auf eine Definition eines Console-Fensters (CON: oder RAW:) mit nachfolgenden Fensterparametern oder mit einem "*" auf das aktuelle Fenster zeigen.

Wenn die Datei nicht geöffnet werden konnte, wird DOSFALSE (null) zurückgeliefert, IoErr() liefert eine genauere Fehlerbeschreibung.

Eingabe(n): name ist ein Zeiger auf einen null terminierten C-String. zugriffsweise long int.

Funktionswert: datei ist ein BCPL-Zeiger auf eine Struktur FileHandle.

Siehe auch Close(), Lock().

Name: **Output**

Aufruf: datei = Output();
BPTR datei;
(struct FileHandle *)BADDR (datei);

Prototyp: BPTR Output(void);

Funktion: Output() gibt einen Datei-Handle auf die Standardausgabedatei zurück, die dem Prozeß beim Start übergeben wurde.

Eingabe(n): keine (void).

Funktionswert: datei ist ein BCPL-Zeiger auf eine Struktur FileHandle.

Name: **ParrentDir**

Aufruf: <D0> lock = ParentDir (<D1> oldlock);
BPTR lock, oldlock;
(struct FileLock *)BADDR (lock);
(struct FileLock *)BADDR (oldlock);

Prototyp: BPTR ParnetDir(BPTR);

Funktion: ParrentDir gibt einen Lock auf das übergeordnete Verzeichnis zurück, wenn ein Lock auf eine Datei oder ein Verzeichnis in diesem Vezeichnis als Parameter angegeben wird.

Wenn man sich im Wurzelverzeichnis der Dateihierarchie einer Disk befindet, liefert ParrentDir den Wert null zurück, der keinen Fehler anzeigt, sondern das Wurzelverzeichnis der Disk repräsentiert, von der der Amiga gebootet wurde.

Eingabe(n): oldlock ist ein BCPL-Zeiger auf eine Struktur FileLock.

Funktionswert: lock ist ein BCPL-Zeiger auf eine Struktur FileLock.

Siehe auch: CurrentDir(), Lock().

Name: **Read**

Aufruf: <D0> gelesene_laenge = Read (<D1> datei, <D2> puffer,
<D2> laenge);
long gelesene_laenge;
BPTR datei;
(struct FileHandle *)BADDR (datei);
char *puffer;
long laenge;

Prototyp: long Read(BPTR, char*, long);

Funktion: Mit einer Kombination aus READ und WRITE können Daten kopiert werden. Read liest Daten aus einer geöffneten Datei in den mit puffer und laenge referenzierten Pufferspeicher. Dabei werden Zeichen gelesen, bis der Puffer ganz gefüllt ist, oder das Dateiende erreicht wird. Es sollte immer überprüft werden, ob der mit laenge übergebene Wert gleich der tatsächliche Größe des Puffers ist. Read meldet zurück, wenn weniger Bytes als angegeben gelesen werden konnten. Der übergebene Wert zeigt die Anzahl der tatsächlich gelesenen Bytes an. Ist gelesene_laenge größer Null, entspricht dieser Wert der Anzahl der gelesenen Bytes. Wird Null übergeben, wurde das Ende der Datei erreicht.

Fehler werden mit dem Return-Code –1 angezeigt. Liest Read von der Tastatur, werden Daten erst dann übergeben, wenn ein <RETURN> eingegeben wurde oder der Puffer voll ist. IoErr() stellt weitere Informationen über einen aufgetretenen Fehler zur Verfügung (zum Beispiel gelesene_laenge = –1).

Eingabe(n): datei ist ein BCPL-Zeiger auf eine Struktur FileHandle.

puffer ist ein Zeiger auf einen Zeichenpuffer, der vorher mit AllocMem() vom Betriebssystem angefordert wurde.

laenge long int.

Funktionswert: gelesene_laenge long int.

Name: **Rename**

Aufruf: <D0> ergebnis = Rename (<D1> alter_name, <D2> neuer_name);

long ergebnis;

char* alter_name, neuer_name;

Prototyp: long Read(char *, char *);

Funktion: Rename versucht, dem mit alter_name bezeichneten Verzeichnis oder Datei den Namen neuer_name zuzuweisen. Existiert bereits ein Verzeichnis oder eine Datei mit dem Namen neuer_name, wird Rename() abgebrochen und ein Fehler mit DOSFALSE gemeldet. alter_name und neuer_name können zusätzlich auch Namen von verschiedenen Verzeichnissen beinhalten. In diesem Fall wird die Datei aus dem alten Verzeichnis entfernt und mit neuem Namen in das neue Verzeichnis kopiert. Das Zielverzeichnis muß zu dieser Aktion bereits bestehen. Beachten Sie bitte, daß Dateien mit Rename() nicht von einer Diskette auf eine andere kopiert werden können.

Eingabe(n): alter_name ist ein Zeiger auf einen null terminierten C-String. neuer_name ist ein Zeiger auf einen null terminierten C-String.

Funktionswert: ergebnis long int.

Name: **Seek**

Aufruf: <D0> alte_position = Seek (<D1> datei, <D2> position, <D3> modus);

long alte_position;

BPTR datei;

(struct FileHandle *)BADDR (datei);

long position;

long modus;

Prototyp: long Seek(BPTR, long, long);

Funktion: Seek setzt den Schreib-Lese-Positionszeiger von datei auf die neue Position position. Ab dieser Position wird dann gelesen oder geschrieben (siehe OPEN und READ). Wurde die Funktion fehlerfrei ausgeführt, liefert der Return-Code die alte_position in der Datei.

Der Rückgabewert –1 zeigt einen Fehler an. Mit IoErr() erhalten Sie weitere Informationen über den aufgetauchten Fehler.

position ist eine Positionsangabe relativ zu einem Basiszeiger in die Datei, den der Anwender durch einen geeigneten Wert für modus wählen kann:

- OFFSET_BEGINNING der Positionszeiger wird relativ zum Dateianfang um position Bytes nach vorne verschoben.
- OFFSET_CURRENT der Positionszeiger wird vorwärts oder rückwärts durch einen positiven oder negativen Wert für position verschoben.
- OFFSET_END der Positionszeiger wird relativ zum Dateiende um position Bytes nach zurück verschoben.

Um an das Dateiende zu gelangen, kann man also Seek() mit position = 1 und modus = OFFSET_END aufrufen.

Mit Seek können Sie leicht Informationen an eine Datei anhängen. Gehen Sie zum Ende der Datei, wie oben beschrieben, und beginnen Sie zu schreiben. Über das Ende einer Datei hinaus kann Seek nicht positionieren.

Eingabe(n): datei ist ein BCPL-Zeiger auf eine Struktur FileHandle.
position long int.
modus long int

Funktionswert: alte_position long int.

Name: **SetComment**

Aufruf: <D0> ergebnis = SetComment (<D1> name, <D2> komment);
long ergebnis;
char *name;
char *kommentar;

Prototyp: long SetComment(char *, char*);

Funktion: SetComment() fügt einen Dateikommentar an eine bestehende Datei an. Der Kommentar darf ein bis zu 80 Zeichen langer null terminierter C-String sein.

Bei einem Fehler gibt die Funktion DOSFALSE zurück, sonst DOSTRUE.

Eingabe(n): name ist ein Zeiger auf einen null terminierten C-String. kommentar ist ein Zeiger auf einen null terminierten C-String.

Funktionswert: ergebnis long int.

Name: **SetProtection**

Aufruf: <D0> ergebnis = SetProtection (<D1> name, <D2-4> schutz_flags);

long ergebnis;

char *name;

long schutz_flags;

Prototyp: long SetProtection(char *, long);

Funktion: SetProtection() setzt oder löscht die Schutzflags einer Datei oder eines Verzeichnisses (siehe PROTECT DOS-Befehle Kapitel 1.2).

Die Schutzflags werden als Bits einer long int Variablen gesetzt:

- FIBF_SCRIPT die Datei ist eine Script-Datei aus DOS-Befehlen.
- FIBF_PURE das Programm ist reentrant und multientrant.
- FIBF_ARCHIVE das Bit wird immer gelöscht wenn die Datei neu beschrieben wird.
- FIBF_READ die Datei kann nicht gelesen werden (wird vom alten Filing-System ignoriert, FFS beachtet die Bits).
- FIBF_WRITE die Datei kann nicht beschrieben werden (wird vom alten Filing-System ignoriert, FFS beachtet die Bits).
- FIBF_EXECUTE die Datei ist ein ausführbares Programm (wird vom DOS ignoriert, die SHELL wertet dieses Bit aus).
- FIBF_DELETE die Datei kann nicht gelöscht werden.

Alle weiteren freien Bits sind für eine spätere Verwendung reserviert und dürfen nicht vom Benutzer verwendet werden.

Um alle benötigten Bits zu setzen, werden die einzelnen Bits mit einem Bitweisen ODER verknüpft.

```
schutz_flags = FIBF_READ | FIBF_WRITE | FIBF_DELETE;
```

Eingabe(n): name ist ein Zeiger auf einen null terminierten C-String. schutz_flags long int.

Funktionswert: ergebnis long int.

Name: **UnLoadSeg**

Aufruf: UnLoadSeg (<D1> segment);

BPTR segment;

Prototyp: void UnLoadSeg(BPTR);

Funktion: Ein mit LoadSeg() geladenes Progamm wird aus dem Speicher entfernt (null ist für segment ein gültiger Wert).

Eingabe(n): segment ist ein BCPL-Zeiger auf den Anfang der Segmentliste des geladenen Programms im Speicher.

Funktionswert: keiner (void).

Name: **UnLock**

Aufruf: UnLock (<D1> lock);
BPTR lock;
(struct FileLock *)BADDR (lock);

Prototyp: void Unlock(BPTR);

Funktion: Ein Dateisystem-Lock, das die Funktionen Lock(), DupLock() oder CreateDir() zurückgeliefert haben, wird freigegeben und gelöscht.

Es ist harmlos lock = 0 zu übergeben.

Eingabe(n): lock ist ein BCPL-Zeiger auf eine Struktur FileLock.

Funktionswert: keiner (void).

Name: **WaitForChar**

Aufruf: <D0> ergebnis = WaitForChar (<D1> datei, <D2> timeout);
long status;
BPTR datei;
(struct FileHandle *)BADDR (datei);
long timeout;

Prototyp: long WaitForChar(BPTR, long);

Funktion: WaitForChar versucht von der Eingabe datei timeout Mikrosekunden lang ein Zeichen zu lesen. Wenn ein Zeichen gelesen werden konnte, gibt WaitForChar() DOSTRUE zurück. WaitForChar darf nur verwendet werden, wenn datei auf ein Console-Fenster weist. Wurde während der angegebenen Zeit kein Buchstabe eingegeben, so wird DOSFALSE zurückgeliefert.

Bugs: Dank eines Fehlers im timer.device der Betriebssystemversionen 1.2 und 1.3 führt ein timeout von null (0) im weiteren zu einem fehlerhaften und unzuverlässigen Verhalten der Diskettenlaufwerke und des timer.device.

Eingabe(n): datei ist ein BCPL-Zeiger auf eine Struktur FileHandle.

timeout long int.

Funktionswert: ergebnis long int.

Name: **Write**

Aufruf: <D0> geschriebene_laenge = Write (<D1> datei, <D2> puffer, <D3> laenge);
long geschriebene_laenge;
BPTR datei;
(struct FileHandle *)BADDR (datei);
char *puffer;
long laenge;

Prototyp: long Write(BPTR, char *, long);

Funktion: Write() schreibt die Anzahl laenge (maximal Größe des Puffers!) Bytes aus dem Zwischenspeicher puffer in die Datei datei und liefert die Anzahl der tatsächlich geschriebenen Bytes zurück. Gab es einen Fehler beim Schreiben, wird der Wert –1 zurückgeliefert. Diese Werte sollten beim Schreiben auf Disketten oder Platten genau untersucht werden, damit keine Daten verloren gehen.

Eingabe(n): datei ist ein BCPL-Zeiger auf eine Struktur FileHandle. puffer ist ein Zeiger auf einen Zwischenspeicher, in dem die Daten stehen, die geschrieben werden sollen, zum Beispiel ein null terminierter C-String. laenge long int.

Funktionswert: geschriebene_laenge long int.

5 Packets

Dieses Kapitel beschreibt den direkten Aufruf von den systeminternen Funktionen des Amiga-DOS.

5.0 Beschreibung der Struktur DosPacket

Über Packets läuft die gesamte Kommunikation zwischen den Prozessen. Ein Packet ist eine Datenstruktur, die auf dem Mechanismus der Datenübertragung zwischen Prozessen, den EXEC zur Verfügung stellt, aufbaut.

Eine EXEC-Nachricht (Message) ist eine in INCLUDE:exec/ports.h deklarierte Struktur, die als einen Bestandteil eine Struktur Node hat, in der ein Feld Name vorhanden ist. Amiga-DOS verwendet dieses Feld als APTR zu einer anderen Struktur Packet. Ein Packet muß Langwort-ausgerichtet sein und hat folgende Struktur:

```
struct DosPacket {
      /* Zeiger auf EXEC Structur Message                        */
     struct Message *dp_Link;
      /* MsgPort für die Antwort mit ReplyMsg() auf das Packet  */
      /* muß vor jedem Verschicken eines Packets neu            */
      /* initialisiert werden, da der Inhalt vom DOS            */
      /* überschrieben wird                                     */
     struct MsgPort *dp_Port;
      /* Konstanten unten                                       */
     LONG dp_Type;
      /* Rückgabewert bei Packets, die DOS-Funktionsaufrufen    */
      /* entsprechen, (gleich Funktionswert)                    */
     LONG dp_Res1;
      /* entspricht Rückgabewert, der mit IoErr() bei           */
      /* DOS-Funktionen abgefragt werden kann                   */
     LONG dp_Res2;
   /*  Übliche Namen bei Packets an Gerätetreiber               */
  #define dp_Action  dp_Type
  #define dp_Status  dp_Res1
  #define dp_Status2 dp_Res2
  #define dp_BufAddr dp_Arg1
```

```
    LONG dp_Arg1;
    LONG dp_Arg2;
    LONG dp_Arg3;
    LONG dp_Arg4;
    LONG dp_Arg5;
    LONG dp_Arg6;
   LONG dp_Arg7;
 }; /* DosPacket */

struct StandardPacket {
    struct Message   sp_Msg;
    struct DosPacket sp_Pkt;
 }; /* StandardPacket */

/* Packet types */
 #define ACTION_NIL                  0
 #define ACTION_GET_BLOCK            2   /* OBSOLETE */
 #define ACTION_SET_MAP              4
 #define ACTION_DIE                  5
 #define ACTION_EVENT                6
 #define ACTION_CURRENT_VOLUME       7
 #define ACTION_LOCATE_OBJECT        8
 #define ACTION_RENAME_DISK          9
 #define ACTION_WRITE               'W'
 #define ACTION_READ                'R'
 #define ACTION_FREE_LOCK           15
 #define ACTION_DELETE_OBJECT       16
 #define ACTION_RENAME_OBJECT       17
 #define ACTION_MORE_CACHE          18
 #define ACTION_COPY_DIR            19
 #define ACTION_WAIT_CHAR           20
 #define ACTION_SET_PROTECT         21
 #define ACTION_CREATE_DIR          22
 #define ACTION_EXAMINE_OBJECT      23
 #define ACTION_EXAMINE_NEXT        24
 #define ACTION_DISK_INFO           25
 #define ACTION_INFO                26
 #define ACTION_FLUSH               27
 #define ACTION_SET_COMMENT         28
 #define ACTION_PARENT              29
 #define ACTION_TIMER               30
 #define ACTION_INHIBIT             31
 #define ACTION_DISK_TYPE           32
 #define ACTION_DISK_CHANGE         33
 #define ACTION_SET_DATE            34
```

```
#define ACTION_SCREEN_MODE          994
#define ACTION_READ_RETURN         1001
#define ACTION_WRITE_RETURN        1002
#define ACTION_SEEK                1008
#define ACTION_FINDUPDATE          1004
#define ACTION_FINDINPUT           1005
#define ACTION_FINDOUTPUT          1006
#define ACTION_END                 1007
#define ACTION_TRUNCATE            1022   /* nur FFS */
#define ACTION_WRITE_PROTECT       1023   /* nur FFS */
```

Das genaue Format eines Packets hängt stark von seinem Typ ab; in allen Fällen aber enthält es einen Zeiger zurück zur Struktur Message, einen Zeiger auf den MsgPort für die Antwort und zwei Ergebnis-Felder.

Sendet Amiga-DOS ein Packet, wird der Zeiger auf den Antwort-MsgPort mit der Adresse des MsgPort des Absenders überschrieben, so daß das Ergebnis an den richtigen Empfänger gesendet werden kann. Deshalb muß der MsgPort für die Antwort an den Absender beim Senden eines Packets an einen Amiga-DOS-Handler-Prozeß vor dem Verschicken des Packets immer korrekt gesetzt werden, wenn eine Struktur erneut verwendet werden soll, die als Antwort zurückgegeben wurde, da Amiga-DOS den Zeiger auf den MsgPort überschreibt. Außer den Feldern, die die Ergebnisse enthalten, die die aufgerufene DOS-interne Funktion zurückliefert, werden alle anderen Felder von Amiga-DOS nicht verändert.

Alle Packets werden an den MsgPort gesandt, der bei der Erzeugung des Prozesses (CreateProc()) eingerichtet wird. Ankommende Nachrichten werden durch Setzen des Signal 8 angezeigt. Wenn dieser Prozeß dann aktiv wird (auf Nachrichten wartende Prozesse warten darauf, daß Signal 8 gesetzt wird), übernimmt GetMsg() die Nachricht vom MsgPort und filtert die Adresse der Struktur Packet heraus. Ist der Prozeß vom Typ Amiga-DOS-Handler-Prozeß, enthält das Packet einen Wert im Feld dp_Type, der eine Aktion anfordert, zum Beispiel Daten lesen. Die Argument-Felder enthalten spezielle Werte, zum Beispiel die Adresse und Größe des Puffers, in den Zeichen geschrieben werden sollen.

Hat der Handler-Prozeß diese angeforderte Aufgabe abgeschlossen, wird die Meldung in der gleichen Struktur Message zum Sender zurückgeschickt, mit der die Aktion angefordert wurde. Der Speicherbereich für die Struktur Message und die Struktur Packet muß vom Versender mit AllocMem() angefordert werden. Der anfragende Prozeß darf diesen Speicherbereich nicht freigeben, bis er vom Handler-Prozeß eine Antwort gegeben hat.

Die freie Verfügung über den Speicherbereich der Nachricht wird dem Empfänger der Nachricht übergeben. Der Versender darf solange den Inhalt der Nachricht nicht verändern, bis er mit ReplyMsg() den Speicherbereich zurückerhalten hat. Dann darf er wieder Daten in ihm ändern.

Normalerweise wird Amiga-DOS vom Empfänger aufgefordert, das Packet zu senden, wenn zum Beispiel eine Aufforderung zum Lesen erteilt wurde. In einigen Fällen jedoch, wenn asynchrone Datenübertragung benötigt wird, kann der auftraggebende Prozeß die Packets zum Handler schicken, wie er es benötigt.

Der Speicherplatz für das Packet und die Struktur Message müssen vom Betriebssystem angefordert werden und dann muß der Zeiger auf den MsgPort des Versenders, in dem die Antwort zurückgegeben wird, initialisiert werden. Der Aufruf von PutMsg() verschickt nun das Packet an seinen Empfänger.

Es ist darauf zu achten, daß die Antwort auf viele abgeschickte Packets entweder an den gleichen MsgPort zurückgeschickt werden kann, von dem die Nachricht abgeschickt wurde, oder an einen anderen beim Verschicken mit angegebenen MsgPort zurückgegeben werden kann.

Unter DOS 1.2 gibt es einen neuen Packet-Action-Typ:

SetFileDate (34). Sie können diesen Packet-Typ dazu verwenden, das Datei- oder Verzeichnis-Datum auf einen bestimmten Wert zu setzen. Das erste Argument ist ein Lock, das zweite ein APTR auf ein Amiga-DOS-Datum, das durch den Aufruf der DateStamp-Routine zurückgegeben wird.

Ein weiterer neuer Packet-Action-Typ ist SetRawMode (994). Dieser neue Modus wird dazu verwendet, das CON:-Device so umzuschalten, daß es genauso reagiert wie das Device RAW:, beziehungsweise um es wieder in den ursprünglichen Zustand zurückzusetzen. Übergeben Sie das TRUE als Argument, um in den RAW-Zustand zu schalten. FALSE stellt den normalen Zustand ein.

Trotzdem existiert ein Unterschied zwischen RAW: und CON: im RAW-Zustand. Sowohl im normalen als auch im RAW-Zustand akzeptiert CON: die Escape-Sequenzen zum Ein- oder Ausschalten der Umwandlung von Linefeed in Carriage Return plus Linefeed. RAW: dagegen formt niemals Linefeeds um (die Sequenzen lauten: CSI 20h zum Einschalten der Konvertierung und CSI 201 zum Ausschalten).

5.1 Packet-Typen

Amiga-DOS unterstützt die im folgenden beschriebenen Packet-Typen. Nicht alle Typen sind für alle Handler zulässig, zum Beispiel sind Zugriffe mit Rename nur für Handler zulässig, die ein Filing-System unterstützen. Für jeden Packet-Typ werden die Argumente und Ergebnisse beschrieben. Der aktuelle Code für jeden Typ erscheint in dezimaler Form unmittelbar neben dem Namen des Symbols. In fast allen Fällen wird ein Fehler durch eine Null im Feld Res1 angezeigt. Dann erscheinen im Feld Res2 zusätzliche Informationen über diesen Fehler. Mit dem Aufruf der Funktion IoErr aus Amiga-DOS werden diese zusätzlichen Informationen ausgegeben.

```
OpenOldFile

Typ   LONG   ACTION_FINDINPUT (1005)
Arg1  BPTR   FileHandle
Arg2  BPRT   Lock
Arg3  BSTR   Name
Res1  LONG   Boolean
```

Die Funktion versucht eine bereits vorhandene Datei für Ein- oder Ausgabe (im vorangehenden Kapitel 3 finden Sie unter Open() mehr Informationen über das Öffnen von Dateien). Um den Lock zu erhalten, müssen Sie DeviceProc() aufrufen, um die Handler-Prozeßkennung festzustellen. IoErr() liefert dann den gewünschten Lock. Alternativ dazu können Sie diese Werte auch direkt aus der Struktur DevInfo lesen. Es ist zu beachten, daß sich der Lock auf das Verzeichnis bezieht, in dem die Datei steht, nicht auf die Datei selbst.

Die Struktur FileHandle muß vom aufrufenden Programm bereitgestellt und initialisiert werden. Dazu werden alle Felder außer fh_Pos und fh_End auf null gesetzt. Die beiden Ausnahmen erhalten den Wert –1. Das Feld fh_Port im Datei-Handle muß den Zeiger auf den MsgPort des Handler-Prozesses beinhalten. Das Ergebnis des Aufrufs ist DOSFALSE, wenn ein Fehler aufgetreten ist. In diesem Fall stehen im Feld Res2 weitere Informationen zu dem aufgetretenen Fehler zur Verfügung. Der Datei-Handle sollte dann gelöscht werden.

```
OpenNewFile

Typ   LONG   ACTION_FINDOUTPUT (1006)
Arg1  BPTR   FileHandle
Arg2  BPTR   Lock
Arg3  BSTR   Name
Res1  LONG   Boolean
```

Die Argumente von Open New File sind gleich denen von Open Old File.

```
Read

Typ   LONG   ACTION_READ ('R')
Arg1  BPTR   FileHandle Arg1
Arg2  APTR   Puffer
Arg3  BSTR   Länge
Res1  LONG   Aktuelle Länge
```

Zum Lesen einer Datei wird der Zeiger auf den MsgPort des Geräte-Handlers aus dem Feld fh_Port des Datei-Handle entnommen. Und der Wert des Feldes fh_Arg1 vom Handle wird in das Feld Arg1 des Packets übernommen. In die beiden anderen Argumentfelder werden die Adresse und Länge des Puffers geschrieben, in den die Daten übertragen werden sollen. Im Feld dp_Res1 steht dann die Anzahl der gelesenen Zeichen. Bei der Erläuterung zur

Funktion Read() finden Sie weitere Informationen zu diesem Thema. Tritt ein Fehler auf, wird bei diesem Aufruf –1 gemeldet, hier stehen im Feld Res2 weitere Informationen zu dem aufgetretenen Fehler zur Verfügung.

```
Write

Typ   LONG    ACTION_WRITE ('W')
Arg1  BPTR    FileHandle Arg1
Arg2  APTR    Puffer
Arg3  BSTR    Länge
Res1  LONG    Aktuelle Länge
```

Die Argumente sind identisch mit denen von Read. Weitere Informationen zum Feld Res1 finden Sie bei der Beschreibung der Funktion Write().

```
Close

Typ    LONG    ACTION_END (1007)
Arg1   BPTR    FileHandle Arg1
Res1   LONG    TRUE
```

Mit diesem Packet wird die geöffnete Datei geschlossen, auf die der Datei-Handle weist. Der MsgPort des Handler-Prozesses ist in der Struktur FileHandle schon eingetragen. Die Funktion hat normalerweise das Ergebnis TRUE. Nachdem eine Datei geschlossen wurde, sollte der dadurch nicht mehr benötigte Speicherplatz für das Datei-Handle wieder freigegeben werden (Free()).

```
Seek

Typ   LONG    ACTION_SEEK (1008)
Arg1  BPTR    FileHandle Arg1
Arg2  LONG    Position
Arg3  LONG    Modus
Res1  LONG    Alte_Position
```

Dieses Packet bewirkt das Gleiche wie die DOS-Funktion Seek(). Als Ergebnis wird die alte Position oder –1 gemeldet, wenn ein Fehler auftritt. Der Zeiger auf den MsgPort des Handler-Prozesses wird vom Datei-Handle übernommen.

```
WaitChar

Typ   LONG    ACTION_WAITCHAR (20)
Arg1  LONG    Timeout
Res1  LONG    Boolean
```

Dieses Packet implementiert die Funktion WaitForChar(). Sie müssen es zum Console-Handler-Prozeß senden, mit der zu wartenden Zeit in Arg1. Das Packet meldet sich zurück, wenn Zeichen zur Übernahme anstehen oder wenn die angegebene Zeit abgelaufen ist. Wird als Ergebnis DOSTRUE übergeben, kann mindestens ein Zeichen von einem nachfolgenden Read gelesen werden.

```
ExamineObject

Typ   LONG    ACTION_EXAMINE_OBJECT (23)
Arg1  BPTR    Lock
Arg2  BSTR    FileInfoBlock
Res1  LONG    Boolean
```

Dieser Packet-Typ implementiert die Funktion Examine(). Er filtert die Addresse des MsgPort des Handlers aus dem Feld fl_Task des Lock. Ist der Lock null, wird der normale Datei-Handler verwendet, der im Feld pr_FileSystemTask der Prozeß-Kontrollstruktur abgelegt ist. Das Ergebnis ist DOSFALSE, wenn ein Fehler auftritt, weitere Informationen über die Fehlerursache stehen dann in dp_Res2. Der FileInfoBlock enthält ansonsten den Inhalt der Felder Name und Kommentar als BSTR.

```
ExamineNext

Typ   LONG    ACTION_EXAMINE_NEXT (24)
Arg1  BPTR    Lock
Arg2  BPTR    FileInfoBlock
Res1  LONG    Boolean
```

Dieser Aufruf implementiert die Funktion ExNext(), die Argumente entsprechen denen von Examine. Beachten Sie, daß der BSTR, der den Dateinamen enthält, zwischen den Aufrufen von ExamineObject und darauffolgenden Aufrufen von ExamineNext nicht gelöscht werden darf, da der BSTR als Zwischenspeicher für DOS-interne Angaben über die Position innerhalb des untersuchten Verzeichnisses verwendet.

```
DiskInfo

Typ   LONG    ACTION_DISK_INFO (25)
Arg1  BPTR    struct InfoData
Res1  LONG    TRUE
```

Mit diesem Packet wird die Funktion Info() realisiert. Normalerweise liefert der Lock zu einem Device auch einen Zeiger auf den MsgPort des Handler-Prozesses. Dieses Packet kann ebenso an einen Console-Handler-Prozeß gesandt werden. In diesem Fall enthält das Feld id_VolumeNode in der Struktur InfoData den Zeiger zu dem Fenster, das der Console-Handler geöffnet hat.

```
Parent

Typ   LONG    ACTION_PARENT (29)
Arg1  BPTR    Lock
Res1  LONG    ParentLock
```

Dieses Packet meldet als Ergebnis den ParentLock auf das übergeordnete Verzeichnis des Verzeichnisses, auf das Lock weist. Es entspricht dem ParentDir()-Funktionsaufruf. Er filtert die Adresse des MsgPort des Handlers aus dem Feld fl_Task des Lock. Ist der Lock null, wird der normale Datei-Handler verwendet, der im Feld pr_FileSystemTask der Prozeß-Kontrollstruktur abgelegt ist.

```
DeleteObject

Typ   LONG    ACTION_DELETE_OBJECT (16)
Arg1  BPTR    Lock
Arg2  BSTR    Name
Res1  LONG    Boolean
```

Dieses Packet implementiert die Funktion Delete(). Es erhält den dazu benötigten Lock vom Aufruf der Funktion IoErr(), der unmittelbar auf einen erfolgreichen Aufruf der Funktion DeviceProc() folgt. Aus DeviceProc() wird der Zeiger auf den MsgPort des Handler-Prozesses übernommen. Der aktuelle Lock gehört zu dem Verzeichnis, in dem die zu löschende Datei abgelegt ist.

```
CreateDir

Typ   LONG    ACTION_CREATE_DIR (22)
Arg1  BPTR    Lock
Arg2  BSTR    Name
Res1  LONG    Lock
```

Dieses Packet implementiert die Funktion CreateDir(). Die Argumente entsprechen denen von DeleteObject. Das Ergebnis ist DOSFALSE (im Fehlerfall) oder der Lock des neuen Verzeichnisses.

```
LocateObject

Typ   LONG    ACTION_LOCATE_OBJECT (8)
Arg1  BPTR    Lock
Arg2  BSTR    Name
Arg3  LONG    Modus
Res1  BPTR    Lock
```

Dieses Packet implementiert die Funktion Lock() und liefert DOSFALSE oder den gewünschten Lock. Die Argumente entsprechen denen von CreateDir, dazu kommt der Modus als Argument 3. Der Modus bezieht sich auf den Typ des Lock, also geteilter oder exklusiver Zugriff.

```
CopyDir

Typ   LONG    ACTION_COPY_DIR (19)
Arg1  BPTR    Lock
Res1  BPTR    Lock
```

Dieses Packet implementiert die Funktion DupLock(). Ist der Lock, der dupliziert werden soll, gleich null, ist auch das Duplikat null. Andernfalls wird der Zeiger auf den MsgPort des Handler-Prozesses aus dem Lock gefiltert und dieser Packet-Typ gesendet. Das Ergebnis ist der neue Lock oder DOSFALSE, wenn ein Fehler aufgetreten ist.

```
FreeLock

Typ   LONG    ACTION_FREE_LOCK (15)
Arg1  BPTR    Lock
Res1  LONG    Boolean
```

Dieser Aufruf implementiert die Funktion UnLock(). Er filtert den Zeiger auf den MsgPort des Handler-Prozesses aus dem Lock. Beachten Sie, daß die Funktion UnLock auf einen Lock mit dem Wert null nichts bewirkt.

```
SetProtect

Typ   LONG    ACTION_SET_PROTECT (21)
Arg1  nicht verwendet
Arg2  BPTR    Lock
Arg3  BSTR    Name
Arg4  LONG    Maske
Res1  LONG    Boolean
```

Dieses Packet implementiert die Funktion SetProtect(). Der Lock ist ein Lock zu dem Verzeichnis, das von DeviceProc() geliefert wird, wie es schon oben bei DeleteObject beschrieben wurde. Die letzten sechs Bit von Maske repräsentieren die Schutzflags, wie sie bei der DOS-Funktion SetProtect() in Kapitel 3 erklärt werden.

```
SetComment

Typ   LONG    ACTION_SET_COMMENT (28)
Arg1  nicht verwendet
Arg2  BPTR    Lock
Arg3  BSTR    Name
Arg4  BSTR    Kommentar
Res1  LONG    Boolean
```

Mit diesem Packet-Typ wird die Funktion SetComment() implementiert. Die Argumente entsprechen denen von SetProtect, lediglich in Arg4 ist der zu vergebende Kommentar als BSTR enthalten.

```
RenameObject

Typ   LONG    ACTION_RENAME_OBJECT (17)
Arg1  BPTR    Alter_Lock
Arg2  BPTR    Alter_Namen
Arg3  BPTR    Neuer_Lock
Arg4  BPTR    Neuer_Namen
Res1  LONG    Boolean
```

Dieses Packet implementiert die Funktion Rename(). Es benötigt dazu einen Lock zum Verzeichnis und den ursprünglichen sowie den neuen Namen. Die Locks für Quell- und Zielverzeichnis werden mit DeviceProc geholt, wie oben bei DeleteObject beschrieben wurde.

```
Inhibit

Typ   LONG    ACTION_INHIBIT (31)
Arg1  LONG    Boolean
Res1  LONG    Boolean
```

Dieses Packet implementiert eine Funktion, die nicht als Amiga-DOS-Funktion zur Verfügung steht. Das Packet enthält einen Wahrheitswert, der besagt, ob die Dateiverwaltung am Starten des Aktualisierungsprozesses für eine neu eingelegte Diskette gehindert werden soll. Ist der Wert TRUE, können Sie Disketten wechseln, ohne daß die Dateiverwaltung den Versuch unternimmt, die neue Diskette zu überprüfen. Während das Wechseln von Disketten für den Computer nicht erkennbar ist, ist der Disketten-Typ mit dem Wert »«Not a DOS disk« (keine DOS-Diskette) belegt, andere Prozesse können dann nicht auf die Diskette zugreifen.

Ist der Wert FALSE, kehrt das System zur normalen Routine zurück, sobald die Aktualisierungsoperation bei der eingelegten Diskette beendet ist. Diese Funktion ist besonders dann sinnvoll, wenn Sie ein Programm wie DISKCOPY schreiben, bei dem viele Diskettenwechsel vorgenommen werden, dabei aber eine Diskette mit unvollständiger Struktur verwendet wird. Nach Aufruf der Funktion bleiben Ihnen die vielen Fehlermeldungen des Disk-Validator erspart, der Sie sonst immer auf die unvollständige Diskette hinweist.

```
RenameDisk

Typ   LONG    ACTION_RENAME_DISK (9)
Arg1  BPTR    Neuer_Name
Res1  LONG    Boolean
```

Auch diese Funktion ist nicht als normaler Amiga-DOS-Funktionsaufruf verfügbar. Das einzige Argument enthält den neuen Namen der Diskette, die gerade im Laufwerk des Handler-Prozesses eingelegt ist, an den das Packet gesandt wurde. Der Diskettenname wird auf der Diskette und in der DOS-Liste im Speicher geändert.

6 Das Amiga-DOS-File-System

Dieses Kapitel beschreibt das Aufzeichnungsformat, das Amiga-DOS beim Beschreiben von Disketten verwendet.

6.1 Die Amiga-DOS-File-Struktur

Der Amiga-DOS-Datei-Handler behandelt Disketten, die mit einer einheitlichen Blocklänge formatiert sind. Verzeichnisse sind mit unbegrenzter Verschachtelungstiefe möglich, dabei kann jedes Verzeichnis Dateien und Unterverzeichnisse enthalten. Als Verschachtelungsart steht eine Baum-Struktur zur Verfügung.

Innerhalb dieser Struktur können Sie nahezu alle hardwaremäßig erzeugten Fehler auf der Diskette beseitigen. Die Arbeit damit erleichtert Ihnen ein Disk-Monitor (zum Beispiel DISKED oder Discovery). Dafür benötigen Sie jedoch genaues Wissen um die Aufzeichnungs-Struktur von Amiga-DOS. Die nachfolgenden Abschnitte beschreiben die möglichen Block-Typen einer Diskette.

6.1.1 Das Wurzelverzeichnis einer Diskette (Root-Block)

Die Wurzel des Dateien-Baumes ist der ROOT-Block, der erste erstellte Block auf der Diskette. Dieser ROOT-Block ist nahezu identisch zu allen anderen Unterverzeichnissen, hat jedoch kein übergeordnetes Verzeichnis (ach was...), und auch der sekundäre Dateityp unterscheidet ihn von allen anderen Blöcken. Im Namen-Feld dieses ROOT-Blocks wird der Name der Diskette eingetragen. Jeder Block einer Diskette enthält eine Check-Summe, die Check-Summe aller Worte in diesem Block ist Null.

Die folgende Abbildung zeigt den Aufbau des ROOT-Blockes (Wurzelverzeichnis):

```
 0    T_SHORT = 2              ; Primärtyp
 1    Header Key               ; Wert ist immer null (0)
 2    Highest Sequence Num     ; Wert ist immer null (0)
 3    Größe der Hashtabelle    ; 72
 4                             ; 0
 5    CHECKSUM                 ; Prüfsumme des Blocks
 6    Anfang des hash table    ;
      ...
77    Ende des hash table      ;
```

```
 78     Bitmap flag              ; null wenn Bitmap ungültig ist
 79     Anfang Bitmap pages      ; Zeiger auf die Blöcke der BM
        ...
104     Ende Bitmap pages        ;
105     Days                     ; Datum letzte Root
106     Mins                     ; Minuten
107     Ticks                    ; 1/50 Sekunden
108     Anfang Diskname          ; BSTR maximal 30 Zeichen
        ...
117     Ende Diskname            ;
118     [Days]                   ; Datum des letzten Schreibzugriffs
119     [Mins]                   ;
120     [Ticks]                  ;
121     Createdays               ; Formatierungsdatum
122     Createmins               ;
123     Createticks              ;
124     nächster Eintrag dieser hash-Kette (immer null)
125     übergeordnetes Verzeichnis (immer null)
126     Erweiterungsfeld (immer null)
127     ST_ROOT = 1              ; Sekundärtyp
```

6.1.2 Benutzerverzeichnisblock

```
  0     T_SHORT = 2              ; Primärtyp
  1     Header Key               ; Wert ist OWN_KEY
  2     Highest Sequence Num     ; Wert ist immer null (0)
  3     0                        ;
  4     0                        ;
  5     CHECKSUM                 ; Prüfsumme des Blocks
  6     Anfang des hash table    ;
        ...
 77     Ende des hash table      ;
 78     0                        ; reserviert
 79     0                        ; reserviert
 80     Protection Bits          ; siehe SetProtection() Kapitel3
 81     0                        ; unbenutzt
 82     Anfang Comment           ; BSTR Kommentar max 80 Zeichen
        ...
104     Ende Comment             ;
105     Days                     ; Datum letzte Änderung
106     Mins                     ; Minuten
107     Ticks                    ; 1/50 Sekunden
108     Anfang Verzeichnisname   ; BSTR maximal 30 Zeichen
        ...
```

```
123     Ende Verzeichnisname     ;
124     nächster Eintrag dieser hash-Kette
125     übergeordnetes Verzeichnis
126     Erweiterungsfeld (immer null)
127     ST_USERDIR = 2           ; Sekundärtyp
```

Benutzerverzeichnisblocks tragen den Typ T_SHORT und als Sekundärtyp die Bezeichnung ST.USERDIR. Die ersten sechs Worte enthalten die Blocknummer und die Größe der Hash-Tabelle. Die 50 Worte am Ende eines Blocks enthalten das Datum und die Zeit der Erstellung des Verzeichnisses, einen Zeiger auf die nächste Datei oder Verzeichnis und einen Zeiger auf das übergeordnete Verzeichnis.

Um eine Datei oder ein Verzeichnis zu finden, wenden Sie zunächst auf dessen Namen eine Hash-Funktion an. Diese Funktion liefert einen Offset in die Hash-Tabelle. Dort befindet sich ein Zeiger auf den ersten Block einer Kette von Blöcken, deren Name denselben Hash-Wert ergibt. Amiga-DOS liest dann den Block ein, vergleicht dessen Namen mit dem gesuchten Namen und geht zum nächsten Block der Kette, bis der gesuchte Name gefunden ist.

6.1.3 Der Dateikopf-Block

```
  0     T_SHORT = 2              ; Primärtyp
  1     Header Key               ; Wert ist OWN_KEY
  2     Blockcount               ; Anzahl benutzte Datenblöcke
  3     0                        ;
  4     first data               ; Zeiger erster Datenblock
  5     CHECKSUM                 ; Prüfsumme des Blocks
  6     Data block72             ; falls vorhanden
        ...                      ; Liste der Datenblockzeiger
 77     Data block 1             ; (rückwärts verwaltet)
 78     0                        ; reserviert
 79     0                        ; reserviert
 80     Protection Bits          ; siehe SetProtection() Kapitel3
 81     Byte Size                ; Dateigröße in Bytes
 82     Anfang Comment           ; BSTR Kommentar max 80 Zeichen
        ...
104     Ende Comment             ;
105     Days                     ; Datum letzte Änderung
106     Mins                     ; Minuten
107     Ticks                    ; 1/50 Sekunden
108     Anfang Dateiname         ; BSTR maximal 30 Zeichen
        ...
123     Ende Dateiname           ;
124     nächster Eintrag dieser hash-Kette
125     übergeordnetes Verzeichnis
126     Extension                ; Zeiger erster Erweiterungsblock
127     ST_FILE = -3             ; Sekundärtyp
```

Jede Datei beginnt mit einem Dateikopf-Block des Typs T_SHORT und des Sekundär-Typs ST_FILE. Anfang und Ende eines Blocks enthalten den Namen, die Zeit der Erstellung und alle Informationen wie ein Verzeichnisblock. Der Hauptteil der Datei besteht aus den Datenblöcken, die von eins an aufwärts numeriert sind. Amiga-DOS speichert die Adressen dieser Blocks in aufeinanderfolgenden Worten ab Wort 6 (in absteigender Reihenfolge, Anfang Wort 77) im File-Header-Block. Normalerweise benutzt Amiga-DOS nicht alle Speicherzellen für diese Liste, und der letzte Datenblock wird auch nicht voll genutzt.

6.1.4 Der Datei-List-Block

Enthält eine Datei mehr Blöcke, als die Blockliste im Dateikopf-Block aufnehmen kann, wird in das Extension-Feld ein Zeiger auf die nächste von dieser Datei verwandte Blockliste geschrieben.

```
  0     T_LIST = 16              ; Primärtyp
  1     Header Key               ; Wert ist OWN_KEY
  2     Block Count              ; benutzte Einträge in Blockliste
  3     0                        ;
  4     0                        ;
  5     CHECKSUM                 ; Prüfsumme des Blocks
  6     Data block N+72          ; falls vorhanden
        ...                      ; Liste der Datenblockzeiger
 77     Data block N+ 1          ; (rückwärts verwaltet)
 78     Anfang Info              ; unbenutzt
        ...
123     Ende Info                ; unbenutzt
124     0                        ;
125     Zeiger auf zugehörigen Fileheader
126     Extension                ; Zeiger nächster Erweiterungsblock
127     ST_FILE = -3             ; Sekundärtyp
```

Der Aufbau eines Datei-List-Blocks ist nahezu identisch mit dem des Dateikopf-Blocks. Datei-List-Blocks gibt es zu jeder Datei so viele, wie nötig sind, um alle Blockadressen der Datenblöcke dieser Datei aufzunehmen. Sie werden über das Feld Extension miteinander verkettet.

6.1.5 Der Datenblock

```
  0     T_DATA = 8               ; Primärtyp
  1     Header Key               ; zeigt auf den Dateikopf
  2     Sequence Num             ; laufende Nummer des Datenblocks
  3     Data Size                ; Anzahl Datenbytes dieser Block
  4     Next Data                ; Zeiger nächster Datenblock
```

```
  5     CHECKSUM                ; Prüfsumme des Blocks
  6     Anfang Data             ; maximal 488 Datenbytes
        ...                     ;
127     Ende Data               ;
```

Datenblöcke enthalten lediglich sechs Informationen zur Dateiverwaltung, jede in einem Wort gespeichert. Die restlichen Worte des Blocks enthalten die eigentlichen Daten, die in der jeweiligen Datei abgespeichert sind. Alle Datenblöcke einer Datei werden mit diesen Daten gefüllt (Anzahl der Worte: Blockgröße –6). Lediglich der letzte kann freien Speicherplatz enthalten. Dessen Next-Data-Block enthält dann null als Zeichen dafür, daß kein weiterer Block existiert.

6.1.6 Die Bitmap

```
  0     Checksum                ; Prüfsumme des Blocks
  1       33 ...    2           ; Blockbelegung (LSB nach MSB)
  2       65 ...   34           ; Blockbelegung
        ...                     ;
 55     1759 ... 1730           ; Blockbelegung (Bits 0 Bis 29)
 56     unbenutzt
        ...
127     unbenutzt
```

6.1.7 Der Bootblock (Diskettentyp und Bootvorgang)

```
  0     Bootblock ID            ; immer BBID_DOS oder BBID_KICK
  1     Checksum                ; Prüfsumme (nur Bootblock)
  2     DOSBLOCK                ; Zeiger auf Wurzelverzeichnis
  3     Boot Code               ; Bootroutine
        ...                     ; beliebt bei Virusprogrammierern
        ...                     ; GrrRRRRrrr!i!i@@1i!

BBID_DOS  = { 'D', 'O', 'S', '\0' }
BBID_KICK = { 'K', 'I', 'C', 'K' }
```

7 Terminal-Ein- und -Ausgabe auf dem Amiga

In diesem Anhang stehen die Zeichen <CSI> für »Control Sequence Introducer«. Zur Ausgabe können Sie [Esc]+[[oder $9B (hex) verwenden, bei der Eingabe wird $9B gemeldet.

7.0 Einführung

Dieser Anhang beschreibt verschiedene Wege, um das Terminal (Tastatur und Bildschirm) für Ein- und Ausgaben zu verwenden. Dieses Terminal kann wie eine normale Amiga-DOS-Datei (mit *, CON: und RAW:) angesprochen werden. Hier nun die Vor- und Nachteile der verschiedenen Methoden:

*: Der Stern erzeugt kein neues Fenster, sondern nutzt das aktive Console-Fenster der SHELL. Nicht der gesamte Zeichensatz steht zur Verfügung. Möglich sind Groß- und Kleinbuchstaben (a–z und A–Z), ASCII-Symbole und Control-Zeichen. Jedes Zeichen, das ein Fernschreiber mit einer Taste erzeugen kann, wird ebenfalls dargestellt. Zusätzlich können alle anderen Zeichen verwendet werden, bei denen das höchste Bit gesetzt ist. ($80–$FF). Die Zeichen [Backspace] zum Löschen eines Zeichens und [Ctrl]+[X] zum Löschen einer Zeile werden ebenfalls unterstützt. Jede <CSI>-Sequenz kann die [Ctrl]-Zeichen C, D, E, F, H und X enthalten. Jedes [CarriageReturn] oder [Ctrl]+[M] wird in [Ctrl]+[J] (neue Zeile) umgewandelt.

CON: Es wird ein neues Console-Fenster definiert, das die gleichen Eigenschaften wie ein SHELL-Fenster hat.

RAW: Der einfache Fall: Mit RAW: (verglichen mit CON:) verlieren Sie die Möglichkeit, den in einer Zeile eingegebenen Text noch zu verändern, bevor er dem von RAW: lesenden Prozeß übergeben wird. Davon sind die Cursor-Tasten ebenso betroffen wie die [Ctrl]-Tasten. Alle Steuerzeichen werden als Zeichen übernommen. Dieser Modus ist sinnvoll, wenn der Programmierer zum Beispiel einen Bildschirmorientierten Editor schreiben will. Er kann dann alle Editier- und Scroll-Funktionen selbst implementieren.

Der komplizierte Fall: Bei Eingabe zusätzlicher Befehle an den Console-Prozessor können Sie weit mehr Informationen erhalten. Zum Beispiel können Sie Informationen über einen Tastendruck oder Maus-Klick erhalten. Weitere Informationen dazu im Abschnitt Auswahl von RAW:-Ereignissen später in diesem Kapitel.

console.device: Mit der direkten Nutzung des console.device hat der Programmierer die weitestgehende Kontrolle über ein virtuelles Terminal. Die Tastenbelegung kann zum Beispiel geändert werden. Die Programmierung des console.device wird hier jedoch nicht genau beschrieben. Informationen hierüber finden Sie unter anderem im Rom Kernel Manual »Libraries und Devices« dessen überarbeitete englischsprachige Neuauflage für die Version 1.3 des Betriebssystems bei Addison-Wesley im Laufe 1989 erscheint (sonst nehmen Sie halt die alte Ausgabe ...).

7.0.1 Hilfreiche Amiga-DOS-Befehle

Zwei sehr hilfreiche Amiga-DOS-Befehle sollen Sie zu eigenen Experimenten anregen:

```
TYPE RAW:10/10/100/30/ HEX
```

übernimmt Eingaben aus einem neuen RAW:-Fenster und zeigt die ASCII-Werte der gedrückten Tasten als Hexadezimalzahlen an. Dies ist der einfachste Weg, um zu überprüfen, welche Zeichen von der Tastatur gesendet werden. Wenn der Zeilenpuffer vollgeschrieben ist, wird die Zeile im SHELL-Fenster angezeigt. Um diesen Befehl abzubrechen muß nur [Ctrl]+[C] eingegeben und der Zeichenpuffer gefüllt werden, damit der TYPE-Befehl terminiert.

Und der Befehl

```
COPY "RAW:10/10/100/30/RAW Eingabe"
"RAW:100/10/200/100/RAW Ausgabe"
```

erlaubt, Zeichen in das (nicht aktive) Eingabefenster zu schreiben und die Zeichen im (aktiven) Ausgabefenster darzustellen. COPY erkennt die Ende-der-Datei-Markierung der RAW:-Eingabe nicht. [Ctrl]+[C] bricht diesen Befehl ab.

Mit dem Hilfsprogramm KEYTOY2000 von der Extras-Diskette kann die Tastaturbelegung angeschaut werden.

7.1 CON:-Tastatur-Eingabe

Wenn Daten von CON: gelesen werden, werden die Eingaben von der Tastatur zwischengespeichert. Man erhält ASCII-Zeichen wie »B» des nationalen Zeichensatzes, der mit SETMAP aktiviert wurde. Die meisten Programme, die Eingaben von der Tastatur erwarten, lesen aus CON:. Einige, zum Beispiel Textverarbeitungs-Programme oder Programme, die die Tastatur als Klaviatur verwenden, lesen jedoch aus RAW:, damit sie die unveränderten Zeichen direkt erhalten und bearbeiten können.

Um die Sonderzeichen auf der Tastatur zu erhalten, muß zusätzlich zur normalen Taste eine der [Alt]-Tasten gedrückt werden. Sie setzen das High-Bit des ASCII-Codes der Taste. Nur das Zeichen Umlaut-Ypsilon (ÿ) mit der Wert $FF (Dezimal 255) ist ein Sonderfall. Den Regeln folgend würde die Kombination [Alt]-[Del] den Wert $FF liefern, die System-

0	$00	NUL	64	$40	@	128	$80		192	$C0	A
1	$01	SOH	65	$41	A	129	$81		193	$C1	Á
2	$02	STX	66	$42	B	130	$82		194	$C2	A
3	$03	ETX	67	$43	C	131	$83		195	$C3	Ã
4	$04	EOT	68	$44	D	132	$84		196	$C4	Ä
5	$05	ENQ	69	$45	E	133	$85		197	$C5	Å
6	$06	ACK	70	$46	F	134	$86		198	$C6	Æ
7	$07	BEL	71	$47	G	135	$87		199	$C7	Ç
8	$08	BS	72	$48	H	136	$88		200	$C8	È
9	$09	HT	73	$49	I	137	$89		201	$C9	É
10	$0A	LF	74	$4A	J	138	$8A		202	$CA	È
11	$0B	VT	75	$4B	K	139	$8B		203	$CB	Ë
12	$0C	FF	76	$4C	L	140	$8C		204	$CC	Ì
13	$0D	CR	77	$4D	M	141	$8D		205	$CD	Í
14	$0E	SO	78	$4E	N	142	$8E	SS2	206	$CE	Î
15	$0F	SI	79	$4F	O	143	$8F	SS3	207	$CF	Ï
16	$10	DLE	80	$50	P	144	$90	DCS	208	$D0	Ð
17	$11	DC1	81	$51	Q	145	$91		209	$D1	Ñ
18	$12	DC2	82	$52	R	146	$92		210	$D2	Ò
19	$13	DC3	83	$53	S	147	$93		211	$D3	Ó
20	$14	DC4	84	$54	T	148	$94		212	$D4	Ô
21	$15	NAK	85	$55	U	149	$95		213	$D5	Õ
22	$16	SYN	86	$56	V	150	$96		214	$D6	Ö
23	$17	ETB	87	$57	W	151	$97		215	$D7	×
24	$18	CAN	88	$58	X	152	$98		216	$D8	Ø
25	$19	EM	89	$59	Y	153	$99		217	$D9	Ù
26	$1A	SUB	90	$5A	Z	154	$9A		218	$DA	Ú
27	$1B	ESC	91	$5B	[	155	$9B	CSI	219	$DB	Û
28	$1C	FS	92	$5C	\	156	$9C	ST	220	$DC	Ü
29	$1D	GS	93	$5D	]	157	$9D	OSC	221	$DD	Ý
30	$1E	RS	94	$5E	^	158	$9E	PM	222	$DE	Þ
31	$1F	US	95	$5F	_	159	$9F	APC	223	$DF	ß
32	$20	SPC	96	$60	`	160	$A0	NBSP	224	$E0	à
33	$21	!	97	$61	a	161	$A1	¡	225	$E1	á
34	$22	"	98	$62	b	162	$A2	¢	226	$E2	â
35	$23	#	99	$63	c	163	$A3	£	227	$E3	ã
36	$24	$	100	$64	d	164	$A4	¤	228	$E4	ä
37	$25	%	101	$65	e	165	$A5	¥	229	$E5	å
38	$26	&	102	$66	f	166	$A6	¦	230	$E6	æ
39	$27	/	103	$67	g	167	$A7	§	231	$E7	ç
40	$28	(	104	$68	h	168	$A8	¨	232	$E8	è
41	$29	)	105	$69	i	169	$A9	©	233	$E9	é
42	$2A	=	106	$6A	j	170	$AA	ª	234	$EA	ê
43	$2B	?	107	$6B	k	171	$AB	«	235	$EB	ë
44	$2C	'	108	$6C	l	172	$AC	¬	236	$EC	ì
45	$2D	-	109	$6D	m	173	$AD	-	237	$ED	í
46	$2E	.	110	$6E	n	174	$AE	®	238	$EE	î
47	$2F	/	111	$6F	o	175	$AF	¯	239	$EF	ï
48	$30	0	112	$70	p	176	$B0	°	240	$F0	ð
49	$31	1	113	$71	q	177	$B1	±	241	$F1	ñ
50	$32	2	114	$72	r	178	$B2	²	242	$F2	ò
51	$33	3	115	$73	s	179	$B3	³	243	$F3	ó
52	$34	4	116	$74	t	180	$B4	´	244	$F4	ô
53	$35	5	117	$75	u	181	$B5	µ	245	$F5	õ
54	$36	6	118	$76	v	182	$B6	¶	246	$F6	ö
55	$37	7	119	$77	w	183	$B7	·	247	$F7	÷
56	$38	8	120	$78	x	184	$B8	¸	248	$F8	ø
57	$39	9	121	$79	y	185	$B9	¹	249	$F9	ù
58	$3A	:	122	$7A	z	186	$BA	º	250	$FA	ú
59	$3B	;	123	$7B	{	187	$BB	»	251	$FB	û
60	$3C	<	124	$7C	|	188	$BC	¼	252	$FC	ü
61	$3D	=	125	$7D	}	189	$BD	½	253	$FD	ý
62	$3E	>	126	$7E	~	190	$BE	¾	254	$FE	þ
63	$3F	?	127	$7F	DEL	191	$BF	¿	255	$FF	ÿ

Tabelle A.1 Der ASCII Zeichensatz

entwickler entschieden sich aus prinzipiellen ästhetischen Gründen dagegen, daß die alternative Belegung eines nicht druckbaren Zeichens (Del) ein Alphazeichen sein soll und änderten für diese Ausnahme den normalen Alt-Modus.

In Tabelle A.1 werden alle Zeichen gezeigt, die auf dem Amiga dargestellt weren können. Die Zeichen NBSP (nicht trennendes Leerzeichen) und SHY (optionaler Trennstrich) werden nur in Textverarbeitungs-Programmen eingesetzt.

7.2 CON:-Bildschirm-Ausgabe

Die Bildschirm-Ausgabe mit CON: ist ähnlich der mit RAW:, jedoch wird das Zeichen <LF> (hex $0A) in das Zeichen für neue Zeile (Carriage Return) umgewandelt. Der Cursor springt also immer in die erste Spalte einer neuen Zeile, wenn das Zeichen <LF> erscheint.

7.3 RAW:-Bildschirm-Ausgabe

Die im Standard ANSI x3.64 definierten Codes werden unterstützt: Diese Steuerzeichen können von Hand eingetippt werden, oder mit Write() auf eine vorher geöffnete Datei geschrieben werden, die das RAW:-Fenster repräsentiert.

Unabhängige Kontroll-Codes (keine anderen Tasten werden vorher betätigt):

Ctrl	Hex	Name	Definition	Beschreibung
H	08	BS	BACKSPACE	Bewegt den Cursor ein Zeichen nach links
I	09	TAB	TAB	Bewegt den Cursor ein Zeichen nach rechts
J	0A	LF	LINE FEED	Zeilenvorschub
K	0B	VT	VERTICAL TAB	Bewegt den Cursor eine Zeile nach unten und scrollt wenn nötig
L	0C	FF	FORM FEED	Löscht den Bildschirm
M	0D	CR	CARRIAGE RETURN	Springt zur ersten Spalte

Ctrl	Hex	Name	Definition	Beschreibung
N	0E	SO	SHIFT OUT	Setzt MSB für jedes Zeichen vor Ausgabe
O	0F	SI	SHIFT IN	Widerruft SHIFT OUT
[	1B	Esc	ESCAPE	
Siehe unten				

Vor den folgenden Zeichen muß Esc betätigt werden:

Zeichen	Name	Definition	Beschreibung
c	RIS	RESET TO INITIAL STATE	Reset des Systems

Vor den folgenden Zeichen muß Esc oder die Tastenkombination Ctrl-Alt und der angegebene Buchstabe gedrückt werden, um die Aktion auszuführen:

Zeichen	Hex	Name	Beschreibung
IND	845tD	INDEX:	Eine Zeile nach unten
E	85	NEL	Nächste Zeile
M	8D	RI	Eine Zeile nach oben
[	9B	CSI	
CONTROL SEQUENCE INTRODUCER:			siehe nächste Liste

Es folgen nun Kontroll-Sequenzen (mit vorangestelltem <CSI>) mit Parametern. Das erste Zeichen in der folgenden Tabelle (unter dem Zeichen <CSI>) zeigt die Anzahl der erlaubten Parameter an. Dabei gilt:

»0« keine Parameter erlaubt
»1« kein oder ein numerischer Parameter
»2« zwei numerische Parameter ("14;94")
»3« eine beliebige Anzahl numerischer Parameter, getrennt durch ein Semikolon (;)
»4« genau 4 numerische Parameter
»8« genau 8 numerische Parameter

<CSI>	Name	Definition	Beschreibung
1 @	ICH	INSERT CHARACTER	Setzt ein oder mehrere Leerzeichen ein, der Rest der Zeile wird nach rechts verschoben
1 A	CUU	CURSOR UP	Cursor nach oben
1 B	CUD	CURSOR DOWN	Cursor nach unten
1 C	CUF	CURSOR FORWARD	Cursor nach rechts
1 D	CUB	CURSOR BACKWARD	Cursor nach links
1 E	CNL	CURSOR NEXT LINE	n Zeilen abwärts in die 1. Spalte
1 F	CPL	CURSOR PRECEDING LINE	n Zeilen aufwärts in die 1. Spalte
2 H	CUP	CURSOR POSITION	»<CSI>Zeile,SpalteH«
1 J	ED	ERASE IN DISPLAY	Lösche auf dem Bildschirm (nur zum Ende der Ausgabe)
1 K	EL	ERASE IN LINE	Lösche in der Zeile (nur zum EOL)
1 L	IL	INSERT LINE	Fügt eine Zeile vor der Zeile mit dem Cursor ein
1 M	DL	DELETE LINE	Löscht die aktuelle Zeile. Bewegt alle nachfolgenden Zeilen um eine nach oben, löscht die letzte Zeile
1 P	DCH	DELETE CHARACTER	Löscht Zeichen
2 R	CPR	CURSOR POSITION REPORT	(nur im Lese-Strom)Format des Reports: »<CSI>Zeile; SpalteR«
1 S	SU	SCROLL UP	Entfernt erste Zeile vom Bildschirm, bewegt alle anderen eine Zeile nach oben, löscht letzte Zeile

<CSI>	Name	Definition	Beschreibung
1 T	SD	SCROLL DOWN	Entfernt letzte Zeile auf dem Bildschirm, bewegt alle anderen eine Zeile nach unten, löscht erste Zeile
3 h	SM	SET MODE	<CSI>20h weist RAW: an, [LF] als »neue Zeile« zu interpretieren.
3 1	RM	RESET MODE	<CSI>201 widerruft SET MODE 20
3 m	SGR	SELECT GRAPHIC RENDITION	Schaltet Grafik-Modus ein.
1 n	DSR	DEVICE STATUS REPORT	

Die folgenden Steuercodes sind keine ANSI-Standard-Sequenzen, sondern werden nur im Amiga verwendet:

<CSI>	Name	Definition	Beschreibung
1 t	aSLPP	SET PAGE LENGTH	Stellt Papierlänge ein
1 u	aSLL	SET LINE LENGTH	Stellt Zeilenlänge ein
1 x	aSLO	SET LEFT OFFSET	Setzt linken Versatz
1 y	aSTO	SET TOP OFFSET	Setzt oberen Versatz
3 {	aSRE	SET RAW EVENTS	Setzt RAW:-Ereignisse abfrage
8 |	aIER	INPUT EVENT REPORT	Report über Eingabe-Ereignisse (bei Lese-Strom)
3 }	aRRE	RESET RAW EVENTS	Abschalten der RAW-Ereignisse
1 ~	aSKR	SPECIAL KEY REPORT	Report über Sondertasten (bei Lese-Strom)

<CSI>	Name	Definition	Beschreibung
1 p	aSCR	SET CURSOR RENDITION	Schaltet Cursor ein, [Esc] [p] schaltet den Cursor aus
0 q	aWSR	WINDOW STATUS REQUEST	Fenster-Status ausgeben
4 r	aWBR	WINDOW BOUNDS REPORT	Größe eines Fensters (bei Lese-Strom)

Beispiele:

Cursor ein Zeichen nach rechts setzen:
<CSI>C oder [TAB] oder <CSI>1C

Cursor 20 Zeichen nach rechts setzen:
<CSI>20C

Cursor in die linke obere Ecke des Bildschirms
(Home Position) setzen:
<CSI>H oder <CSI>1;1H oder <CSI>1;H

Cursor in die vierte Spalte der ersten Zeile des Fensters setzen:
<CSI>1;4H oder <CSI>;4H

Bildschirm löschen:

```
[FF] oder [Ctrl]-L          ;Zeichen »Clear Screen« oder
<CSI>H<CSI>J                ;»Home« und Rest des Bildschirms
                             löschen oder
<CSI>H<CSI>23M              ;»Home« und 23 Zeilen löschen oder
<CSI>1;1H<CSI>23L           ;»Home« und 23 Zeilen einfügen
```

7.4 RAW:-Tastatur-Eingabe

Lese-Zugriffe auf eine RAW:-Datei ergeben einen Byte-Strom nach dem Standard ANSI x3.64. Dieser Strom enthält normale Zeichen und RAW:-Ereignis-Informationen. Diese können mit den SET RAW EVENTS (aSRE) und RESET RAW EVENTS (aRRE) Kontroll-Sequenzen wie unten beschrieben angefordert werden.

Haben Sie eine RAW:-Eingabe angefordert, wenn kein Datenfluß stattfindet, wartet der Rechner, bis wieder Daten zur Verfügung gestellt werden. Mit der Funktion »WaitForChar« kann Dateneingang abgefragt werden.

Bei normaler Einstellung werden von den Funktionstasten und Cursortasten folgende Zeichen zurückgeliefert:

Taste	ohne Shift	mit Shift
[F1]	<CSI>0~	<CSI>10~
[F2]	<CSI>1~	<CSI>11~
[F3]	<CSI>2~	<CSI>12~
[F4]	<CSI>3~	<CSI>13~
[F5]	<CSI>4~	<CSI>14~
[F6]	<CSI>5~	<CSI>15~
[F7]	<CSI>6~	<CSI>16~
[F8]	<CSI>7~	<CSI>17~
[F9]	<CSI>8~	<CSI>18~
[F10]	<CSI>9~	<CSI>19~
[HELP]	<CSI>?~	<CSI>?~
(identisch mit und ohne Shift)		

Cursortasten:

Taste	ohne Shift	mit Shift
[Auf]	<CSI>A	<CSI>T~
[Ab]	<CSI>B	<CSI>S~
[Links]	<CSI>C	<CSI> A~
(beachten Sie das Leerzeichen)		
[Rechts]	<CSI>D	<CSI> @~
(beachten Sie das Leerzeichen)		

7.5 Auswahl von RAW:-Ereignissen

Wenn RAW: genutzt wird, erhält man die ANSI-Daten und die gerade beschriebenen zusätzlichen Kontroll-Codes. Wenn man mehr Informationen über bestimmte Eingabe-Ereignisse braucht, können diese vom Console Driver angefordert werden.

Müssen Sie zum Beispiel wissen, wann welche Taste gedrückt und wieder losgelassen wurde, müssen Sie RAW:-Tastatur-Eingabe verlangen. Das geschieht mit Ausgabe von

```
<CSI>1{
```

an das Terminal.

RAW:-Eingabe-Ereignisse:

Im RAW:-Tastatureingabemodus erhält man bei Tastendrücken nicht mehr nur ein einzelnes Zeichen, sondern einen Datensatz der Form:

```
<CSI>1;0;<tastencode>;<indikator>;0;0;<sekunden>;<microsekunden>|
```

Nummer	Beschreibung
0	Keine Operation; wird intern genutzt
1	RAW:-Tastatur-Eingabe
2	RAW:-Maus-Eingabe
3	Ereignis; wird immer dann gesendet, wenn ein Fenster aktiviert wird.
4	Pointer-Position
5	(unbenutzt)
6	Timer
7	Gadget gedrückt
8	Gadget losgelassen
9	Requester aktiv
10	Menü-Nummer
11	Schließ-Gadget
12	Fenstergröße verändert
13	Fenster neu gezeichnet
14	Preferences verändert
15	Disk entnommen
16	Disk eingelegt

Wenn eines dieser Ereignisse angefordert wird, erhält man weitere Informationen in der folgenden Form:

<CSI><Gruppe>
<Untergruppe>
<Tastencode>
<Indikator>
<X>
<Y>
<Sekunden>
<Microsekunden>

<CSI>	ist ein Ein-Byte-Feld. Es ist immer gleich $9B hexadezimal.
<Gruppe>	ist der RAW:-Eingabe-Ereignis-Code aus obenstehender Tabelle.
<Untergruppe>	ist nicht benutzt und immer gleich Null.
<Tastencode>	zeigt an, welche Taste gedrückt wurde (Bild A.1 und Tabelle A.2 zeigen die Nummern). Mit diesem Feld wird auch die Maus abgefragt.
<Indikator>	zeigt den Status von Tastatur und System. Dieses Feld ist wie folgt definiert:

Bit	Maske	Taste
0	0001	[SHIFT] links
1	0002	[SHIFT] rechts
2	0004	[CAPS LOCK] * Sondertaste, siehe unten
3	0008	[CTRL]
4	0010	[Alt] links
5	0020	[Alt] rechts
6	0040	linke Amiga-Taste gedrückt
7	0080	rechte Amiga-Taste gedrückt
8	0100	10er-Block
9	0200	Wiederholung
10	0400	Interrupt; nicht benutzt
11	0800	mehrfach ausgegeben; dieses (das aktive) oder alle Fenster
12	1000	linker Maus-Knopf
13	2000	rechter Maus-Knopf
14	4000	mittlerer Maus-Knopf; an der normalen Amiga-Maus nicht vorhanden
15	8000	Maus relativ; angegebene Mauskoordinaten sind relativ und nicht absolut

Die [CAPS LOCK]-Taste wird besonders behandelt. Sie liefert nur einen Tastencode, wenn sie gedrückt wird, nicht jedoch, wenn sie wieder losgelassen wird. Das entsprechende Bit, das Großbuchstaben kennzeichnet (80 hex), wird jedoch bei den nachfolgenden Tastendrücken korrekt gesetzt. Drücken Sie die [SHIFT]-Taste, so daß die LED aufleuchtet, wird der Tastencode 62 ([CAPS LOCK] gedrückt) gesendet. Drücken Sie nun wieder die Taste und die Leuchtdiode geht aus, wird 190 ([CAPS LOCK] losgelassen) gesendet. Die Tastatur meldet diese Taste also so lange als gedrückt, bis sie ein zweites Mal gedrückt wird.

Die Felder <Sekunden> und <Microsekunden> enthalten die Zeit, zu der das Ereignis stattfand, im Format der Systemzeit. Diese wird als Langwort gespeichert.

Hierzu ein Beispiel: Drücken Sie die Taste »B« und lassen Sie sie wieder los, während die linke Umstelltaste und die rechte Amiga-Taste gedrückt waren, erhalten Sie folgende Informationen über RAW:

```
<CSI>1;0;35;129;0;0;23987;99|
<CSI>1;0;163;129;0;0;24003;18|
```

Die beiden aufeinanderfolgenden Felder mit dem Inhalt »0« werden bei Tastatur-Eingaben nicht benutzt. Sie dienen zum Auslesen der Maus-Koordinaten und liefern die X- und Y-Koordinaten des Mauszeigers.

Das Feld <tastencode> enthält den Dezimalwert des ASCII-Codes der Taste, die gedrückt wurde. Zählen Sie 128 dazu, erhalten Sie den Code der losgelassenen Taste.

In der nachfolgenden Tabelle finden Sie:

- Den Wert, den CON: für einen bestimmte RAW:-Tastencode liefert, wenn eine bestimmte Taste gedrückt wurde.
- Die Belegung der deutschen Standard-Tastatur.

RAW:-Tasten-Code	Wert ohne Shift	Wert mit Shift
00	[	]
01	1	!
02	2	"
03	3	#
04	4	$
05	5	%
06	6	&
07	7	/
08	8	(
09	9	)
0A	0	=
0B	ß	?
0C	«	'
0D	\	|
0E	nicht definiert	
0F	0	0 (Zehnerblock)
10	Q	q
11	W	w
12	E	e
13	R	r
14	T	t
15	Z	z
16	U	u
17	I	i
18	O	o
19	P	p
1A	Ü	ü
1B	+	*
1C	nicht definiert	
1D	1	1 (Zehnerblock)
1E	2	2 (Zehnerblock)
1F	3	3 (Zehnerblock)
20	A	a
21	S	s
22	D	d
23	F	f

RAW:- Tasten- Code	Wert ohne Shift	Wert mit Shift
24	G	g
25	H	h
26	J	j
27	K	k
28	L	l
29	Ö	ö
2A	Ä	ä
2B	#	^
2C	nichtdefiniert	
2D	4	4 (Zehnerblock)
2E	5	5 (Zehnerblock)
2F	6	6 (Zehnerblock)
30	<	>
31	Y	y
32	X	x
33	C	c
34	V	v
35	B	b
36	N	n
37	M	m
38	, (Komma)	;
39	. (Punkt)	:
3A	- (Bindestrich)	_ (Unterstrich)
3B	nicht definiert	
3C	. (Punkt)	. (Zehnerblock)
3D	7	7 (Zehnerblock)
3E	8	8 (Zehnerblock)
3F	9	9 (Zehnerblock)
40	Leertaste	
41	Backspace	
42	Tab	
43	Enter	
44	Return	
45	(Esc)	
46	Del	
47	nicht definiert	
48	nicht definiert	
49	nicht definiert	
4A	-	- (Zehnerblock)
4B	nicht definiert	

RAW:- Tasten- Code	Wert ohne Shift	Wert mit Shift
4C	[Cursor nach oben]	Scroll nach unten
4D	[Cursor nach unten]	Scroll nach oben
4E	[Cursor vorwärts]	Scroll nach links
4F	[Cursor rückwärts]	Scroll nach rechts
50	[F1]	<CSI>10~
51	[F2]	<CSI>11~
52	[F3]	<CSI>12~
53	[F4]	<CSI>13~
54	[F5]	<CSI>14~
55	[F6]	<CSI>15~
56	[F7]	<CSI>16~
57	[F8]	<CSI>17~
58	[F9]	<CSI>18~
59	[F10]	<CSI>19~
5A	[	{ (Zehnerblock)
5B	]	} (Zehnerblock)
5C	/	/ (Zehnerblock)
5D	*	* (Zehnerblock)
5E	+	+ (Zehnerblock)

Beim alten Amiga 1000 sind die Codes 5A bis 5E nicht belegt, da auf der Tastatur entsprechende Tasten nicht vorhanden sind.

5F	[Help]
60	[Linke Shift]
61	[Rechte Shift]
62	[CAPS LOCK]
63	[Ctrl]
64	linke [Alt]-Taste
65	rechte [Alt]-Taste
66	[Linke Amiga]
67	[Rechte Amiga]
68	linke Maustaste <nicht übernommen>
69	rechte Maustaste <nicht übernommen>
6A	mittlere Maustaste <nicht übernommen>
68 - 6A	werden im Moment von Intuition nur geliefert, wenn die Maus an den GamePort 1 angeschlossen ist.

6B	nicht definiert
6C	nicht definiert
6D	nicht definiert
6E	nicht definiert
6F	nicht definiert
70-7F	nicht definiert
80-F8	obenstehende Tasten nicht gedrückt oder bereits losgelassen. 80 für 00, F8 für 7F.
F9	letzter Key-Code war ungültig.
FA	Tastaturpuffer voll
FB	nicht definiert; reserviert für »Ausfall des Tastaturprozessors«
FC	Selbsttest der Tastatur fehlgeschlagen.
FD	Power-up Key Stream Start. Wurden während des Einschaltvorgangs Tasten gedrückt, werden diese zwischen FD und FE gesendet.
FE	Power-up Key Stream Ende.
FF	nicht definiert
FF	Mausaktion, nur im Moment der Aktion. <nicht übernommen>

Tabelle A.2: Tastaturcodes

Hinweise zur obenstehenden Tabelle:

1. nicht definiert bedeutet, daß diese Zahl bei normaler Tastaturbelegung nicht erzeugt wird. Ändern Sie jedoch mit SetMap diese Belegung, kann dieser Wert jedoch, je nach Tastaturbelegung, erzeugt werden.
2. <nicht übernommen> bezieht sich auf die Abfrage der Maus-Tasten. Mit der Sequenz <CSI>2{ weisen Sie den Console-Handler an, diese Mausklicks zu melden. Geben Sie dies nicht gesondert an, werden Mausklicks über RAW: nicht gemeldet.

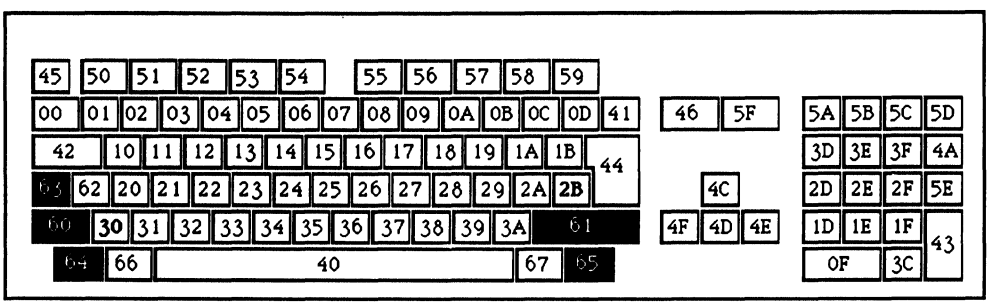


Bild 7.1: RAW und CON

Anhang Listings

CHILDPROC.C

```
; /*
lc -cusfa -v -j73 -M -O childproc
blink from childproc.o to childproc lib lib:lc.lib
quit
*/

/* Netter kleiner Mißbrauch der Regeln für Script-Dateien (HiHi!) */
/* Um das Programm zu kompilieren und linken kann es als         */
/* Script-Datei benutzt werden.                                  */
/* Geben Sie dazu auf der SHELL folgenden Befehl ein:            */
/* EXECUTE childproc.c                                           */
/* -j73 schaltet die Warnung aus, die das Semikolon am Dateianfang*/
/* bemängelt.                                                    */
/* -v schaltet das Stackchecking aus                             */

/*****************************************************************/
/* Modul:          childproc.c                                   */
/*---------------------------------------------------------------*/
/* Autor:          Wilfried Häring                               */
/* Date:           19.4.1989                                     */
/* Aenderungen:    22.4.1989 aufgeräumt und druckfertig gemacht  */
/*---------------------------------------------------------------*/
/* Beschreibung:                                                 */
/* Sohnprozeß wird vom Beispielprogramm gestartet, verschickt    */
/* Antwort und terminiert.                                       */
/* Übrigens:                                                     */
/* Wie immer, so auch hier:                                      */
/*           all rights reserved - alle Rechte vorbehalten       */
/*****************************************************************/

#ifdef AZTEC_C
/* haha - pack ein!                                              */
/* Bei Aztec ging das Kompilieren schon immer schneller...       */
/* kleiner (geklauter) Scherz am Rande                           */
```

```
#else

/* Sie haben den richtigen Compiler                                  */

#include <proto/exec.h>
#include <proto/dos.h>
#include <exec/types.h>
#include <exec/ports.h>
#include <exec/tasks.h>
#include <exec/alerts.h>
#include <exec/execbase.h>
#include <libraries/dos.h>
#include <libraries/dosextens.h>
#include <stdio.h>
#include <string.h>
#include "proclaunch.h"


/* Die einzig immer konstante Adresse im ganzen Betriebssystem    */
#define ABSEXECBASE ((struct ExecBase **)4L)

/* Mindestens benötigte Version der DOS-Library                    */
#define DOS_REV0
/* Definition des Console-Fensters des Sohnprozesses               */
#define WINDOW "CON:20/20/320/80/childproc"

struct ExecBase**SysBase;
struct DosLibrary*DOSBase;

/* Lattice Schlüsselwort __saveds deklariert eine Funktion, bei    */
/* deren Aufruf das Register A4 zwischengespeichert, und auf das   */
/* Standarddatensegment gesetz wird. Das ist für Interrupt-Handler*/
/* und für Funktionen, die mit AddTask() oder wie in diesem
   Beispiel*/
/* mit LoadSeg() und CreateTask() als paralleler Task (Prozeß)     */
/* gestartet werden, unbedingt notwendig, außer es wird absolute   */
/* Adressierung (Compileroption -b0) verwendet.                    */

void __saveds childproc ( void )
{
struct MsgPort  *child_port;    /* aus Prozeßstruktur des Sohnes */
struct MsgPort  *new_port;      /* wird neu angelegt */
struct Process  *child_proc;    /* Prozeß-Kontrollstruktur */
Infopacket      *rcv_cmd;       /* Empfangene Nachricht */
BPTR            stdio;          /* i/o-handle - nice to have BCPL */
int             result;         /* Rückgabewert */
int             exit;           /* Schleifenterminator */
```

```
/* Wenn wie bei diesem Beispiel ohne Standard-Startup-Code gelinkt*/
/* wird, muß der C-Programmierer SysBase und DOSBase selbst       */
/* initialisieren.                                                */
SysBase = ABSEXECBASE;

result = FAIL;

  /* Prozeß-Kontrollstruktur des eigenen Prozesses finden !       */
child_proc = (struct Process *)FindTask ( 0 );
  /* Zeiger auf MsgPort in der eigenen Prozeß-Kontrollstruktur
     holen*/
child_port = (struct MsgPort *) &(child_proc->pr_MsgPort);

  /* Zum Synchronisieren auf Startnachricht des Vaterprozeß warten*/
WaitPort ( child_port );
rcv_cmd = (Infopacket *)GetMsg ( child_port );

  if ( DOSBase = (struct DosLibrary *)
                              OpenLibrary ( "dos.library",DOS_REV ) )
  {
    result = ERROR;

     /* Console fuer i/o oeffnen                                  */
    if ( stdio = Open ( WINDOW ,MODE_NEWFILE ) )
    {
      result = WARN;

       /* Eigenen privaten MsgPort öffnen, um den Standard-MsgPort*/
       /* in der Prozeß-Kontrollstruktur für das DOS freizuhalten.*/
      if ( new_port = CreatePort ( NULL, 0L ) )
      {
        result = ALLRIGHT;

         /* Bis hierher hat alles geklappt, OK an Vaterprozeß     */
        rcv_cmd->flag = result;
        rcv_cmd->x_data = new_port;
        ReplyMsg ( (struct Message *)rcv_cmd );

         /* Ankommende Nachrichten einsammeln und auswerten       */
        exit = FALSE;
        while ( !exit )
        {
          WaitPort ( new_port ); /* remember: no busy-loops
          !!!!!!!! */
          while ( !exit &&
                   ( rcv_cmd = (Infopacket *)GetMsg ( new_port ) ) )
          {
```

```
            switch ( rcv_cmd->flag )
            {
            case CMD_REQUEST :
               /* Beispiel für normale Nachricht irgendwas zu tun */
              Write ( stdio, rcv_cmd->x_data,
                                 (long)strlen ( rcv_cmd->x_data ) );
              Write ( stdio, "\n", 1 );
              rcv_cmd->flag = ALLRIGHT;
              break;

            case CMD_KILL :
               /* Achtung, der Vaterprozeß will uns killen        */
              Write ( stdio, rcv_cmd->x_data,
                                 (long)strlen ( rcv_cmd->x_data ) );
              Write ( stdio, "\n", 1 );
              rcv_cmd->flag = ALLRIGHT;
              exit = TRUE;          /* Austieg aus der event-loop */
              break;

            default :
               /* Unerwartete Nachricht erhalten, kann damit
                  nichts */
               /* anfangen!                                       */
              Write ( stdio, "never\n", 6L );
              rcv_cmd->flag = WARN;
              break;
            }
             /* Wenn der Prozeß gekillt werden soll, muß er vor   */
             /* der Antwort an den Vater erst komplett aufgeräumt */
             /* haben     */
            if ( !exit )
              ReplyMsg ( (struct Message *)rcv_cmd );
          }
        }
        Delay ( 200 );                    /* vier Sekunden warten */

         /* Jetzt wird alles wieder korrekt aufgeräumt !          */
        DeletePort ( new_port );
      }
      Close ( stdio );
    }
    CloseLibrary ( (struct Library *)DOSBase );
  }
```

```
    /* Wenn ein Fehler beim Öffnen der Resourcen geschah, wird ein  /
    /* Fehlercode zurückgegeben, wenn der Vaterprozeß den Sohn     */
    /* killen will, sind jetzt alle Resourcen aufgeräumt.          */
   if ( (result != ALLRIGHT) || exit )
   {
     rcv_cmd->flag = result;
     ReplyMsg ( (struct Message *)rcv_cmd );
   }

   Wait ( 0 );/* auf das Gelöscht werden oder den Weltuntergang */
              /* warten */
}

#endif
```

PROCLAUNCH.C

```
; /*
lc -cusfa -L -j73 -O -M proclaunch
quit
*/

/* Netter kleiner Mißbrauch der Regeln für Script-Dateien (HiHi!) */
/* Um das Programm zu kompilieren und linken kann es als         */
/* Script-Datei benutzt werden.                                  */
/* Geben Sie dazu auf der SHELL folgenden Befehl ein:            */
/* EXECUTE proclaunch.c                                          */
/* -j73 schaltet die Warnung aus, die das Semikolon am Dateianfang*/
/* bemängelt.                                                    */
/* -v schaltet das Stackchecking aus                             */

/*****************************************************************/
/* Modul:          mainproc.c                                    */
/*---------------------------------------------------------------*/
/* Autor:          Wilfried Häring                               */
/* Date:           18.4.1989                                     */
/* Aenderungen:    22.4.1989 aufgeräumt und druckfertig gemacht  */
/*---------------------------------------------------------------*/
/* Beschreibung:                                                 */
/* Beispielprogramm startet einen Sohnprozeß, wartet auf         */
/* Antwort und terminiert.                                       */
/* Übrigens:                                                     */
/* Wie immer, so auch hier:                                      */
/*          all rights reserved - alle Rechte vorbehalten        */
/*****************************************************************/

#ifdef AZTEC_C
```

```
/* haha - pack ein!                                                    */
/* Bei Aztec ging das Kompilieren schon immer schneller...             */
/* kleiner (geklauter) Scherz am Rande                                 */

#else

/* Sie haben den richtigen Compiler                                    */

#include <proto/exec.h>
#include <proto/dos.h>
#include <exec/types.h>
#include <exec/ports.h>
#include <exec/tasks.h>
#include <exec/memory.h>
#include <libraries/dos.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "proclaunch.h"

#define CHILD_PRI      0                    /* Standardeinstellung */
#define CHILD_STACK    4000               /* Standardeinstellung */

void main ( int argc, char **argv )
{
struct MsgPort    *father_port;  /* MsgPort für Antwort an Sohn   */
struct MsgPort    *child_port;   /* aus Prozeßstruktur des Sohnes */
Infopacket        *send_cmd;     /* Abgeschickte Nachricht        */
Infopacket        *rcv_cmd;      /* Antwort des Sohnprozesses     */
/* Prozeß-Kontrollstruktur des Sohnprozesses                       */
struct Process    *child_proc;
BPTR              child_segment; /* Segmentliste des Sohnes       */
int               result;        /* Rückgabewert                  */

  result = FAIL;

   /* MsgPort erzeugen, in dem Antworten des Sohnes empfangen     */
   /*  werden */
  if ( father_port = CreatePort ( NULL, 0L ) )
  {
    result = ERROR;

     /* Code des Sohnprozesses laden                               */
    if ( child_segment = LoadSeg ( CHILDPROC ) )
    {
       /* Sohnprozeß initialisieren und starten                    */
      if ( child_port = CreateProc ( CHILDPROC,
```

```
                                    CHILD_PRI,
                                    child_segment,
                                    CHILD_STACK ) )
{
  result = WARN;

   /* Zeiger auf Prozeß-Kontrollstruktur des Sohnprozesses  */
   /* holen                                                 */
  child_proc = (struct rocess *)
            ( ( (long)child_port ) - sizeof ( struct Task ) );

   /* Speicher für die Nachrichten an den Sohnprozeß vom    */
   /* Betriebssystem anfordern                              */
  if ( send_cmd =
AllocMem ( sizeof ( Infopacket ), MEMF_PUBLIC | MEMF_CLEAR ) )
  {
     /* Nachrichtenstruktur initialisieren                 */
    send_cmd->msg.mn_Node.ln_Type = NT_MESSAGE;
     /* Adressat der Antwort                               */
    send_cmd->msg.mn_ReplyPort = father_port;
    send_cmd->msg.mn_Length = sizeof ( *send_cmd );

    printf ( "Schicke Nachricht an Sohn!\n" );

     /* Startnachricht über Standard-MsgPort in der Prozeß- */
     /* Kontrollstruktur an den Sohnprozeß schicken         */
    send_cmd->flag = CMD_HELLO;
     /* Abschicken und auf Antwort warten                  */
    PutMsg ( child_port, (struct Message *)send_cmd );
     /* Niemals busy-wait !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! */
    WaitPort ( father_port );
     /* Antwort abholen, aber keine Antwort mehr           */
     /* zurückschicken */
    rcv_cmd = (Infopacket *)GetMsg ( father_port );

    if ( rcv_cmd->flag == ALLRIGHT )
    {
      result = ALLRIGHT;

      printf ( "Sohnprozeß ist in Ordnung!\n" );
       /* über universales Datenfeld Informationen abholen  */
      child_port = rcv_cmd->x_data;        /* neuer MsgPort */

      /* Jetzt können beliebig Daten an den neuen MsgPort   */
      /* geschickt werden.                                  */
```

```
          printf ( "Nachfrage an Sohnprozeß geschickt\n" );
          send_cmd->flag = CMD_REQUEST;
          send_cmd->x_data = "Hello son, hope you're fine!";
          PutMsg ( child_port, (struct Message *)send_cmd );
          WaitPort ( father_port );
          rcv_cmd = (Infopacket *)GetMsg ( father_port );

          if ( rcv_cmd->flag == ALLRIGHT )
            printf ( "Antwort über privaten, neuen MsgPort!\n" );
          else
            printf ( "Sohnprozeß meldet Fehler!\n" );

          printf ( "Sohnprozeß soll sich auf sein Ende
                    vorbereiten!\n" );
          send_cmd->flag = CMD_KILL;
          send_cmd->x_data = "Prepare to die!";
          PutMsg ( child_port, (struct Message *)send_cmd );
          WaitPort ( father_port );
          rcv_cmd = (Infopacket *)GetMsg ( father_port );

          if ( rcv_cmd->flag == ALLRIGHT )
            printf ( "Sohnprozeß sieht dem Ende gefaßt ins
                      Auge!\n" );
          else
            printf ( "Da ging was schief!\n" );

        }
        else
          printf ( "Kein Speicher für das Infopacket!\n" );

         /* Und ab jetzt alles wieder sauber aufräumen !!!!!!!! */
        FreeMem ( send_cmd, sizeof ( Infopacket ) );
      }

      DeleteTask ( (struct Task *)child_proc );
    }
    else
      printf ( "Sohnprozeß konnte nicht gestartet werden!\n" );

    UnLoadSeg ( child_segment );
  }
  else
    printf ( "Sohnprozeß konnte nicht geladen werden!\n" );
```

```
    DeletePort ( father_port );
  }
  else
    printf ( "Der Vaterprozeß konnte keinen MsgPort erzeugen!\n" );

  exit ( result );
}

#endif
```

CHILDPROC.H

```
/* Name des Moduls, das als Prozeß gestartet werden soll          */
#define CHILDPROC"childproc"

#define ALLRIGHT    0              /* Rückgabewert wenn alles OK */
#define WARN        5                                /* Warnung */
#define ERROR      10                                 /* Fehler */
#define FAIL       20    /* Da ging so ziemlich alles in die Hose */

/* Flags, die an den Sohnprozeß geschickt werden                  */
#define CMD_HELLO   1                      /* Korrekt gestartet? */
#define CMD_REQUEST 2                       /* Normale Nachricht */
#define CMD_KILL    3     /* Aufräumen und auf das Löschen warten */

typedef struct infopacket
{
  struct Message  msg;         /* Exec braucht diesen Eintrag !!! */
  long            flag;    /* welche Art von Aktion wird erwartet */
  void           *x_data;   /* Zeiger auf beliebige weitere Daten */
} Infopacket;
```

Stichwortverzeichnis

A

Abbruch-Flag 33
absolute 96
ADDBUFFERS 56
Adressierungs-Arten 160
Adreßvarianten 161
Aktualisierungsprozeß 35
aktuelle Zeile 224f.,237f.
ALIAS 54
Alink 181ff.
allgemeine Direktiven 172f
Amiga-DOS-Anweisungen 36
Amiga-DOS-Befehle 31ff.
Amiga-DOS-Betriebssystem 21
Amiga-DOS-File-Struktur 354
Antialiasing 95
Argumente 33f.,233
ASCII-String 160
ASK 122
aspect 94
ASSEM 155
Assembleraufruf 155
Assemblerprogrammierung 153
ASSIGN 45f.,57
ATOM 203ff.
ATOM-Anweisung 205
Attention-Flag 59
Audiokanäle 293
Ausgabedateien 249
Ausgabeumleitung 33,42
Autoboot-Festplatte 134
AVAIL 58

B

Batchprogrammierung 32ff.,113
Baud-Rate 99
bedingte Assemblierung 169
Befehlsdateien 248
Befehlsformat 33
Befehlsgruppen 235
Befehls-Macros 54
Befehlsnamen 233
Befehlsprozessor-Nummer 114
Befehlssyntax 233
Befehlszeile 185
Benutzerverzeichnisblock 355
Bibliotheksfunktion 325
Bildschirm-Ausgabe 362
Bildschirm-Editor 209ff.
Bildwiederholfrequenz 93
BINDDRIVERS 58
Bitmap 358
Blitter 293
Blockform 176
Blockkontroll-Befehle 215
BlocksPerTrack 145
Blocksteuerung 215
Bootblock 358
Bootdiskette 136
booten 49,134ff.
Bootpartition 135ff.
BootPri 147
Boot-Sektor 38
Bootvorgang 49
bounded 96
BREAK 59
BSS-Block 176
BSS-Module 182
Buff.ers 146
BufMemType 146

C

cancel 264
cap-word 267
CD 41,59
center 95
chancel 94
change printer 94

CHANGETASKPRI 60
childproc.c 374
Chip-RAM 74,293
CLI-Command 261
clockpointer 61,131
CLONE 63
close 325
CMD 61
CNOP 169
Code-Segmente 198ff.
CON 22f.,52,347,359ff.
console.device 360
Console-Handler 60
Controller A2090A 135
Copper 293
COPY 44,62f
copy-region 262
C-Programm 295
CreateDir 326
CreateProc 326
Cross-Referenz-Tabelle 183
Ctrl-Code 53
CurrentDir 327
Cursor 210
Cursorposition 216
Custom-Chip 292

D

DATE 42,64
Datei löschen 43
– schützen 40
– umbenennen 43
Datei-Handles 320
Dateikopfblock 356
Datei-List-Block 357
Dateinamen 24
Datenblock 357
Datenstrukturen 310ff.
DateStamp 328
Datum stellen 42
DC 166
DCB 166
default 49
delay 328
delet-back 267
DELETE 43,65
delete-blanks 267
DeleteFile 329
delete-forw 267
density 95
Device 28,49,145
DeviceProc 329
DEVS 30
Dienstprogramm 113
DIR 39,66
Directory 18
Direktiven 162
DISKCHANGE 67
DISKCOPY 37,68
DISKDOKTOR 69
Diskette booten 38
– formatieren 37
– kopieren 37
– umbenennen 38
Diskvalidator 78
dithering 95
DMA-Kanäle 292
DosPacket 345
Dos-Prompt 31
DosType 147
DOWNLOAD 70
DPAT 141
DRIVE 35
Drucker 94
Druckertreiber 272ff.
DS 166
DupLock 330

E

ECHO 72
ED 209
ED-Befehle 219ff.
EDIT 209,222,232
edit pointer 94
EDIT-Befehle 238,254ff.
EDIT-Kommando 224
editieren 54,259
Edit-Menü 261ff.
Editor 209ff.
Eingabedateien 248
Eingabe-Umleitung 33
Eingabe-Zeichen 236
Einstellwerte 235
END 165

end-of-buffer 265
end-of-line 266
end-of-window 265
ENDC 170
ENDCLI 45ff.,73
ENDM 171
ENDSKIP 122
EQU 165
EQUR 165
Escape-Sequence 53
EVAL 122
examine 330
Exception-Vektoren 300
EXECUTE 32,114,331
execute-file 270
execute-line 270
execute-macro 269
exit 332
ExNext 333
expand-window 265
externe Referenzen 181
– Symbole 172
Extras-Menü 268

F

Fachausdrücke 49
FAIL 168
FAILAT 32,124
Fast-Filing-System 56,134
Fast-Fonts 76
Fast-RAM 74,293
FASTMEMFIRST 74f.
FAULT 73
Fehlercodes 202
Fehlermeldung 202,207f.
fence-match 268
Festplatten-Controller 134
file handle 49
FILENOTE 76
File-System 145
FIXFONTS 71
Flags 145
Flicker-Fixer 93
FONTS 30
FORMAT 37,77f.,169
formatieren 37
FROM-Datei 34

G

Geräte 28
GETENV 79
globale Datenstrukturen 315ff.
Vektorgröße 108
GlobVec 146
goto-line 265
Grafik-Systemsoftware 94
GRAPHICDUMP 79
Grauwertumsetzung 94

H

Handler 52,144
Handler-Prozeß 310,346
Hardwaresprites 293
Header 173
HighCyl 146
Hintergrundfarbe 53
HiRes-Modus 93
Hunks 182
hunk_break 198
hunk_bss 176
hunk_code 175
hunk_data 175
hunk_debug 180
hunk_end 180
hunk_ext 177
hunk_header 197
hunk_name 175
hunk_overlay 198
Hunk_Overlay_Tabelle 193
hunk_reloc16 177
hunk_reloc32 176
hunk_reloc8 177
hunk_symbol 179
hunk_unit 174

I

ICONX 124
IDNT 173
IF 125ff.,169
IFC 170
IFD 170
IFNC 170
IFND 170
ignore 96

INCLUDE 172
indent 264
INFO 41,80
Info 334
Info-Substruktur 316
Inhaltsverzeichnis 39
INHIBIT 78
InitialHunk 195
INITIALIZE 78
INITPRINTER 81
input 335
input device 290
Input-File 184
insert-buffer 263
insert-file 260
INSTALL 38,81
interaktive Editierung 254
Interlace-Modus 93
Interleave 145
IoErr 334
IsInteractive 335

J

JOIN 82
justify-buffer 263

K

key repeat delay 93
key repeat speed 93
Kickstart 134,293
kill-buffer 263
kill-line 266
kill-region 261
kill-to-eol 266
Klammerzeichen 118
Knoten 183
Kommando 34
Kommandozeilen-Interpreter 22
Konfigurationsdatei 271
kopieren 37,44

L

LAB 128
Label-Eingabe 129
Label-Feld 156
last saved 93
Lattice-C-Compiler 297
Laufwerk 29
left offset 95
LIBS 30
limits 95
Line-Menü 266
line-to-top 267
Linker 178,181ff.
Link-Map 188
LIST 40,82,167
list-buffers 262
Listing-Kontrolle 167
LLEN 168
Loader 181
Load-File 182,196
loadSeg 335
LOADWB 45,86
LOCK 86,321,336
logische Geräte 29f.,49
lokale Label 157
löschen 43
LowCyl 146
lower-region 262
lower-word 267

M

Macro 55,170
MAKEDIR 45,87
Makro-Assembler 153ff.
Makro-Direktiven 170ff.
MAP 188
Mask 146
MASK2 173
MaxTransfer 146
MC68k-Familie 291
MEMACS 258ff.
MERGEMEM 88
MEXIT 172
Mikroprozessor MC 68000 154
Module 182
MORE 88
MOUNT 89,144ff.
Mountlist 69,144ff.
Move-Menü 265
multiply 96
Multitasking 21,287
Mustererkennungsfähigkeit 141

N

NARG 171
new-cli 261
NEWCLI 47,89
NEWCON 22f.,52
NEWSHELL 47,90
next-line 267
next-page 266
next-window 264
next-word 266
next-w-page 265
Nibble 99
NOBOOT 81
Node 183
NOFASTMEM 91
NOFORMAT 169
NOICONS 77
NOL 167
NOLIST 167
NOOBJ 168
NOPAGE 167
Null-String 243

O

Objekt Code 49
Objekt-File 182
Objekt-File-Struktur 173f
OFFSET 164
Offset-Liste 176
one-window 264
Opcode-Feld 157
open 337
open-line 266
Operanden-Feld 158
operationale Fenster 241
Operationen 250
Operatoren 158
output 337
Overlay-Baum 190ff.
Overlay-Direktive 189
Overlays 189

P

Packet 344ff.
Packet-Typen 347
PAGE 167
PAR 61
parallele Schnittstelle 61,99
Parameterliste 52
Parameterübergabe 114
ParrentDir 338
PATH 91
Patternmatching 140
PCD 143
Peripheriegerät 28
physikalische Geräte 28f.
Pixels 96
Platzhalter 114
PLEN 168
PreAlloc 145
Preferences 80,93ff.
previous-line 267
prev-page 266
prev-w-page 265
prev-window 264
prev-word 266
PRINTFILES 96
Priorität 108
Priority 145
proclaunch.c 378
Program Unit 182
Programmablauf 236
Programm-Counter 154
Programmierrichtlinien 295ff.
Programmstatus 252
Programmsteuerung 214
Projekt-Menü 260
Prompt 97,236
PROTECT 40,97
Prozessor-Register 154
Prozeß-Stapelspeicher 312
Prozeßverwaltung 311
Pufferspeicher 56

Q

Qualifikatoren 227
Quellcode 156
Quelldatei 156,246
query-s-r 268
QUIT 128,261
quote-char 264

R

RAD 31,131,137ff.
RAM 31
RAM-Disk 30f.,100
RAM-Speicher 58
RAW 347,359ff.
READ 99,338
read-file 260
redisplay 263
REG 165
rekursive Befehle 108
RELABEL 38,99
Relokation 176,182
REMOVE 57
REMRAD 100
RENAME 43,100,260,339
reserved 145
reset all 93
reset-keys 270
RESIDENT 44,101
residente Libraries 183
Returncode 124
Root-Block 354
RORG 164
Rückgabewert 124
RUN 102,332

S

save 94
save-as-file 260
save-exit 260
save-file 260
save-mod 260
scaling 95
Scanned Libraries 183
Schleifen 250
Schleifendurchlauf 123
Schnittstellentreiber 81
Schriftarten 53
Schwellwert 94
Script-Datei 32
–, Steuerbefehle 122ff.
scroll-down 266
scroll-up 266
search-backward 268
SEARCH 103
search-forward 268
Search-Menü 268
search-replace 268
SECTION 164
seek 339
SeekOffset 195
Segment-Listen 319
select-buffer 263
Semaphore-Mechanismus 301
SER 61
serielle Schnittstelle 61,94,99
set 166,268
set-arg 268
set-key 270
set-mark 262
SETCLOCK 64,104
SetComment 340
SETENV 105
SETMAP 106
SETPATCH 105
SetProtection 341
shade 94
Shell 17,51ff.
Shell-Fenster 22,47
Shell-Prozeß 31
Shell-Server 52
show-line 267
shrink-window 265
SKIP 128f
SKIP-Block 122
smoothing 94
SORT 106
SPAT 140
SPC 167
Speicherverwaltung 319
split-window 264
Sprungziele 128
STACK 52,107
Stacksize 108,146
Stapelspeicher 107
start-macro 269
start-of-line 266
Startup 147
startup-sequence 45,92,129ff.
Status 60,108
Statusflags 301
Status-Register 154

stop-macro 269
stream 49
Suchausdrücke 235
Suchpfad 92
Surfaces 145
Swap-dot&mark 265
switch-case 267
Symbol-Dateneinheit 177
Symbol-Definition 165
Symbole 159
SymbolHunk 196
SymbolOffsetX 196
SYS 30
System-Diskette 49
Systemkonfiguration 93
Systemsoftware 289f
Systemuhr 104

T

Task 327
Tastatur-Eingabe 360,366
temporäre Datei 114
Terminal-Ausgabe 359ff.
Terminal-Eingabe 359ff.
Text 60/80 93
Texteditor 39
Timerbausteine 294
Timesharing 290
top-of-buffer 265
top-of-window 265
Trackdisk-Device 60,71
transpose 264
TTL 168
Turbokarte 291
TYPE 42f.,109

U

umbenennen 38,43
Umgebungsvariable 79
Unit 145
UnLoadSeg 341
UnLock 342
upper-region 262
upper-word 267
use 94

V

verifizieren 252
VERSION 110
Verzeichnis 18,25
– einrichten 45
– wechseln 41
Verzeichnis-Struktur 26
visit-file 260
Volume-Name 49
Vordergrundfarbe 53

W

WAIT 110
WaitForChar 342
Wertzuweisung 118
WHICH 111
WHY 43,111
WINDOW 52
Window-Menü 264ff.
WITH-Files 186
Word-Menü 267
Workbench 18,147
workbench interlace 93
Workbench-Versionen 19
Write 343
Wurzelverzeichnis 354

X

XDEF 172
XREF 172,188

Y

yank 262

Z

Zahlen 160,235
Zeichenketten 234
Zeilen-Editor 209,222f.
Zeilenfenster 241
Zeilennummer 224,237
Zeit stellen 42

Amiga-DOS 1.3

Der Autor:

WILFRIED HÄRING wurde im November 1968 geboren. Er besuchte von 1978 bis zum Sommer 1988 die Oranienschule in Wiesbaden. Anfang der achtziger Jahre entdeckte er sein Interesse an Computern, als er für einige Zeit mit einem PET2001 arbeitete. Seine Computerkarriere führte über den Commodore 64 und einen IBM-PC zu 68000er-Systemen wie Apple Lisa und Macintosh sowie zu Grafikworkstations unter dem Betriebssystem Unix. Als 1986 der Amiga nach lange brodelnder Gerüchteküche der Fachpresse vorgestellt wurde, schlug sein Herz sofort für ihn, und er wurde einer der Amiga-Programmierer der ersten Stunde. Seit September 1988 ist der Autor Mitarbeiter eines Softwarehauses im Bereich Methoden und Werkzeuge. Er entwickelt dort unter anderem grafisch interaktive Benutzerführungen. Außerdem schreibt er als freier Autor in verschiedenen Publikationen. Neben seinem Beruf findet er noch genug Zeit, um sich intensiv mit dem Amiga auseinanderzusetzen.

Mit der neuen Version 1.3 des Amiga-Betriebssystems wurden die Möglichkeiten dieses Rechners noch einmal erheblich verbessert. Die bedienungsfreundliche Shell hat das behäbige CLI ersetzt, das DOS wurde durch einige mächtige Befehle erweitert, und das totgesagte Diskettenverwaltungssystem erwachte durch das FastFileSystem zu neuem Leben. Mit der Möglichkeit, eine Reset-feste RAM-Disk einzurichten, entfallen auch die Zwangs-Kaffeepausen während der Bootphase. Eine Schwierigkeit bleibt: die »neue Vielfalt« und das recht komplizierte, Multitasking-fähige Betriebssystem machen es Einsteigern und Programmierern nicht leicht. Dem abzuhelfen, ist die Intention des Autors.
Das Amiga-DOS 1.3 Anwender- und Programmiererbuch ist zweigeteilt. Der erste Abschnitt ist den Anwendern gewidmet; der Autor erklärt die Shell und die Amiga-DOS-Befehle, erläutert die Batch-Programmierung (vor allem die Möglichkeiten der Startup-Sequence, die Mountlist, den Makroassembler und den Linker ALINK). Schließlich erhalten Sie eine detaillierte Bedienungsanleitung für den Bildschirm-Editor ED, den Zeilen-Editor Edit und für MEMACS. Im Anhang finden Sie eine exakte Beschreibung der Druckertreiber, die sich auf der 1.3-Workbench bzw. Extras-Diskette befinden, und viele nützliche Tips zur Konfiguration Ihres Druckers.
Im zweiten Teil wendet sich der Autor an die Programmierer. Er lehrt Einsteiger, mit dem Betriebssystem und der Hardware umzugehen. Sie erfahren, welche Programmierrichtlinien für den Amiga gelten, um sicherzustellen, daß Ihre Software auch in Zukunft auf neuen Amiga-Geräten läuft. Sämtliche Datenstrukturen, die Amiga-DOS verwendet, sind aufgelistet, ebenso finden Sie eine alphabetisch sortierte Funktionsbibliothek. Die direkte Steuerung von Handler-Prozessen (Packets) und das Aufzeichnungsformat des alten Filing-Systems und des neuen FastFileSystem werden erläutert. Schließlich wird die Terminal-Ein- und -Ausgabe auf dem Amiga erklärt (CON: und RAW:)

Aus dem Inhalt:

Anwenderbuch:

- Erster Kontakt mit Amiga-DOS – die Shell
- Die Amiga-DOS-Befehle
- Batch-Programmierung – eine eigene Startup-Sequence
- Booten von FastFileSystem und von der RAD:
- Die Mountlist
- Der Makroassembler
- Der Linker ALINK
- ED, Edit, und MEMACS

Programmiererbuch:

- Erster Kontakt mit Amiga-DOS für Programmierer
- Programmierrichtlinien für das Programmieren in C
- Die Amiga-DOS-Datenstrukturen
- DOS-Bibliotheksfunktionen
- Packets
- Das Amiga-DOS-File-System
- Terminal-Ein- und -Ausgabe auf dem Amiga

Hardware-Anforderungen:

Amiga 500, 1000 oder 2000 mit mindestens 512 Kbyte Speicherkapazität.

Software-Anforderungen:

Workbench 1.3, Amiga Extras 1.3 sowie Kickstart 1.3 (für A1000-Diskette, für A500/2000-Kickstart-ROM. Ohne Kickstart 1.3 ist das neue Betriebssystem nur eingeschränkt nutzbar).

ISBN 3-89090-802-0

06900

4 001057 908024

Markt&Technik

DM 69,– sFr 66,20 öS 538,–